DYNAMIC GRID-BASED DATA DISTRIBUTION MANAGEMENT IN LARGE-SCALE DISTRIBUTED SIMULATIONS

Amber Joyce Roy, A. B.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2000

APPROVED:

Azzedine Boukerche, Major Professor
Paul Tarau, Committee Member
Stephen R. Tate, Chair of Graduate Studies in Computer Science
Roy T. Jacob, Committee Member and Chair of the Department of Computer Science
C. Neal Tate, Dean of the Robert B. Toulouse School of Graduate Studies

Roy, Amber Joyce, Dynamic Grid-Based Data Distribution Management in Large Scale Distributed Simulations. Master of Science (Computer Science), December 2000, 102 pp., 17 tables, 23 illustrations, references, 31 titles.

Distributed simulation is an enabling concept to support the networked interaction of models and real world elements that are geographically distributed. This technology has brought a new set of challenging problems to solve, such as Data Distribution Management (DDM). The aim of DDM is to limit and control the volume of the data exchanged during a distributed simulation, and reduce the processing requirements of the simulation hosts by relaying events and state information only to those applications that require them. In this thesis, we propose a new DDM scheme, which we refer to as dynamic grid-based DDM. A lightweight UNT-RTI has been developed and implemented to investigate the performance of our DDM scheme. Our results clearly indicate that our scheme is scalable and it significantly reduces both the number of multicast groups used, and the message overhead, when compared to previous grid-based allocation schemes using large-scale and real-world scenarios.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSRATIONS

1. INTRODUCTION

Interactive simulation has played a key role in the remarkable advances that have occurred in recent years in the areas of science and technology. Simulation is *imitating of the operation of a real-world system over time* [2]. Simulation allows the generation of an artificial history of the system, which can then be observed and studied. The behavior of the system can be described and analyzed, leading to inferences concerning operating characteristics of the real world. Therefore, simulation can also be described as a problem solving methodology for the solution of many real-world problems.

One might question the approach of building a *simulation* of a system, for in some circumstances it is certainly easier to observe the real-world system directly. However, simulation is needed to evaluate systems that *difficult* or even *dangerous* to study in the physical world, such as hurricanes or nuclear reactions. One might also wonder whether simulation is necessary, as opposed to modeling, for it seems possible that scientists would be able to study natural, chemical, or other systems analytically using mathematically-based models. Indeed, researchers frequently use computer models. However, many real-world systems cannot be accurately modeled analytically due to their size and complexity or because of a lack of human understanding regarding the precise functioning of the system.

In addition to representing real-world systems, computer simulation can also be used to investigate man-made systems that have not been developed. Thus we can modify our previous definition of simulation as follows: Simulation is imitating of the

operation of a real or *imagined* system over time [13]. For example, suppose city policy makers wished to assess a proposal to replace the stop signs at several intersections with stoplights. A traffic simulation could be built that would evaluate the impact of the new traffic signals, as compared to the stop signs. Another important example of simulation is in the design of VLSI chips, which are tested extensively in a virtual environment before the chips are actually made. Errors discovered by simulating the proposed design are generally easier and less costly to correct than mistakes that are found after the fabrication process is complete.

In this section we have mentioned many uses of computer simulation, including weather and traffic patterns, nuclear reactions, and VLSI circuits. Assessment of communications networks, such as cellular phone channels, is yet another area where simulation is appropriate. In the coming years, simulation will also be used to evaluate new technologies and the latest discoveries.

1.1 Military Simulations

This thesis attacks problems in the application area of military simulations. We shall now discuss some of the most prevalent military applications for simulation. *War gaming simulations* are often used to evaluate different strategies for attacking or defending against an opposing force, or for acquisition decisions to determine the number and type of weapon systems that should be purchased [13]. War gaming simulations are also known as *aggregated simulations*, since they are generally comprised of representations of battalions, divisions, etc., as opposed to individual aircraft and tanks.

On the other hand, *training simulations* are concerned with single platforms and do represent tanks, submarines, and so on. These simulations usually have a *human-in-the-loop*, meaning that ship captains, weapons operators, and other personnel can participate in a virtual environment where they are controlling the military vehicles, ships, aircraft, or other platforms being represented by the computer system. As opposed to arcade or console games that simulate battles for the players' entertainment, military training simulations are designed to prepare users for actual combat situations.

Just as the combat readiness of soldiers, sailors, and pilots is tested and honed using training simulations, *test and evaluation simulations* assess the performance of physical or mechanical components. Existing components, such as radar systems, interface directly with the computer simulation for the purpose of evaluating the performance of the item under various conditions. Proposed devices can also be simulated and then tested in a virtual environment that simulates the actual situation in which the device is to be used.

Among the three types of military applications for simulation that we have discussed, the one we are primarily targeting is training simulations. Specifically, we aim to improve the efficiency of large-scale military training simulations. The reader should keep in mind, however, that our work can also be applied to other application areas such as the entertainment or telecommunications industries.

1.2 Large-Scale Distributed Simulations

This thesis focuses on issues relevant to large-scale distributed simulations. In this section we define and discuss large-scale distributed simulations. We begin by describing the nature of computer simulation. Next we explain what is generally meant by the term "large-scale," and then we illustrate the motivation behind using a *distributed* approach to large-scale simulation.

1.2.1 Large-Scale Simulations

By *large-scale* simulations we refer to simulations that represent a very large number of vehicles, planes, or other simulated entities. There is no agreed upon minimum number of total entities that must be represented before a simulation is considered to be large scale. We believe 1,000 entities is a reasonable lower limit, and this is the number of entities we have used in our simulation experiments that we will discuss in Chapter 6. The upper limit for the number of entities that can be simulated is basically dependent on the simulation and also on the underlying computing resources, and the increase in memory capacity and processor speed that has occurred in recent years has lead to larger and large simulations being performed. For example, a simulation used with the Synthetic Theatre of War (STOW) was designed to support up to 100,000 simulated entities [6], and a simulation of orbiting satellites successfully represented 1,000,000 entities [26].

Our use of the term large-scale does not refer to the scope of the simulation in terms of time or space. Nonetheless, many large-scale training simulations represent

engagements that occur in a large geographic area, such as the Pacific Ocean or even the entire globe, as in the case of the orbiting satellites. Of course, from the perspective of an astrologer who uses simulations to study the Milky Way galaxy, the earth is a very small region. The same may be said regarding the notion of time. To a geologist who uses simulations to predict what the land formations on the earth will look like a million years from now, a few days is an extremely short duration. Military training simulations usually simulate anywhere between a few minutes and several days of combat.

Even for a simulation representing only one hour of one battle, simulating thousands or even millions of entities requires significant computing resources, including memory, disk space, and CPU cycles. In the next section we investigate a powerful technique to handle the computing requirements of large-scale simulations.

1.2.2 Distributed Approach to Simulation

Distributed simulations are run on several networked computers, which together can support much larger simulations that could be conducted on one computer alone [13]. The computers taking part in a distributed simulation are often connected by the same local area network (LAN) and may even be physically located in the same room. However, many distributed simulations can be run on computers that are not on the same LAN and instead are connected through the Internet. In such a scenario, the computers, also knows as *machines*, *hosts*, or *nodes*, participating in the simulation may be geographically distributed across the United States or around the world.

Besides allowing the computing resources of multiple machines to be combined in order to support one large-scale simulation, distributed simulations that can be run over

the Internet provide another advantage. Users in different geographical locations can observer and participate in the same simulation. This ability is especially important for military training simulations, which we identified in the previous section as being the application our work is targeting. Large-scale distributed training simulations that are conducted over the Internet allow military personnel in different locations, and also from different branches of the service, to all participate in a joint training exercise.

There are clear advantages to using a distributed simulation as oppose to using only a single computer to run the simulation. However, a difficulty in the distributed strategy arises because each machine has its own CPU, memory, and internal clock, and the tasks of synchronizing the processes on various machines and coordinating the communication among them is non-trivial. In Chapter 3 we shall discuss the Run-Time Infrastructure (RTI), which is one method of supporting communication and coordination among simulations being run on distributed computers. We have chosen to use the RTI approach, and have developed our own RTI, as well as other related modules that we describe in the following section.

1.3 Architecture Design

In this section we explain the architecture of the software system that we have developed [5]. The components of this framework are described in greater detail in the following chapters, so the present discussion is aimed at giving an overall summary of the structure of our system. Our software framework is comprised of four components.

1) Tank Dogfight Federate – simulates tank entities

2) UNT-RTI – provides services to a federate

3) Grid-Kit – implements our dynamic grid-based DDM approach

4) Fixed Grid Module – implements fixed grid-based DDM approach

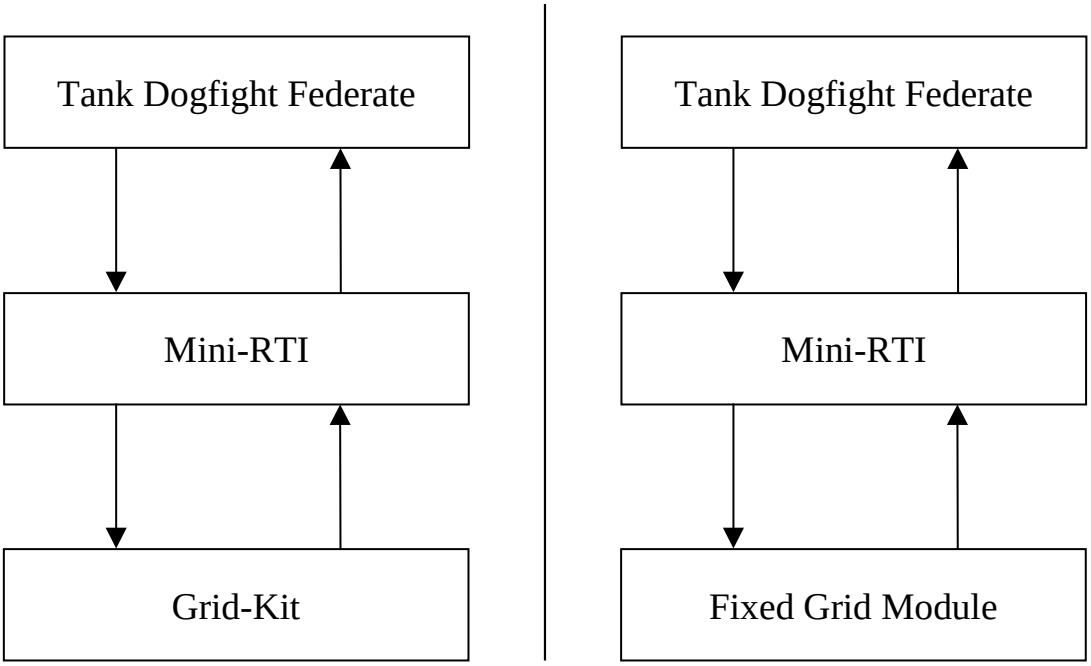

Figure 1. Simulation with Grid-Kit and Fixed Grid Module

The relationship among these modules is shown in Figure 1. The Grid-Kit and Fixed-Grid Module can be interchanged, depending of which type of Data Distribution Management (DDM) scheme is desired. The term federate refers to a simulation. When multiple federates take part in a distributed simulation, these federates are said to combine and form a federation.

1.3.1 Modularity

The modularity of our system has several advantages. The ability to swap out the Grid-Kit and replace it by the Fixed Grid Module, while holding the other components constant, allows us to compare the two approaches of DDM in a scientific manner. For example, we performed simulation experiments using certain parameters for the Tank Dogfight scenario, such as a particular number of tank entities, using the Grid-Kit configuration shown on the left in Figure 1. Then we used the same parameters and performed the experiment using the Fixed Grid Module setup, as illustrated on the right in Figure 1. Finally, we compared the results obtained using the Grid-Kit versus the Fixed Grid Module.

Another benefit of the modularity of our system is that the Grid-Kit can be used with other RTIs. We have integrated our Grid-Kit with our UNT-RTI, but the Grid-Kit can also be used to build new RTIs or to enhance existing ones. For example, our Grid-Kit could be used with the TM-Kit developed at Georgia Tech [14] to develop a new RTI. An additional advantage of our modular system is that another federate can replace the Tank Dogfight federate. Our Grid-Kit can be used with different federates and various RTIs, and this flexibility was one of the driving factors in our architecture design.

1.3.2 Peer-to-Peer

Modularity was an important objective in our architecture design, as was the scalability of our system. It is well know that centralized coordinators or managers can hamper the performance of a software system, as compared to a more distributed

approach. For this reason, our system uses a peer-to-peer paradigm. There is no central server or database. Just as federates in a federation are running on *N* different computers, our UNT-RTI and Grid-Kit components are running on all *N* machines. The peer-to-peer concept is illustrated in Figure 2.

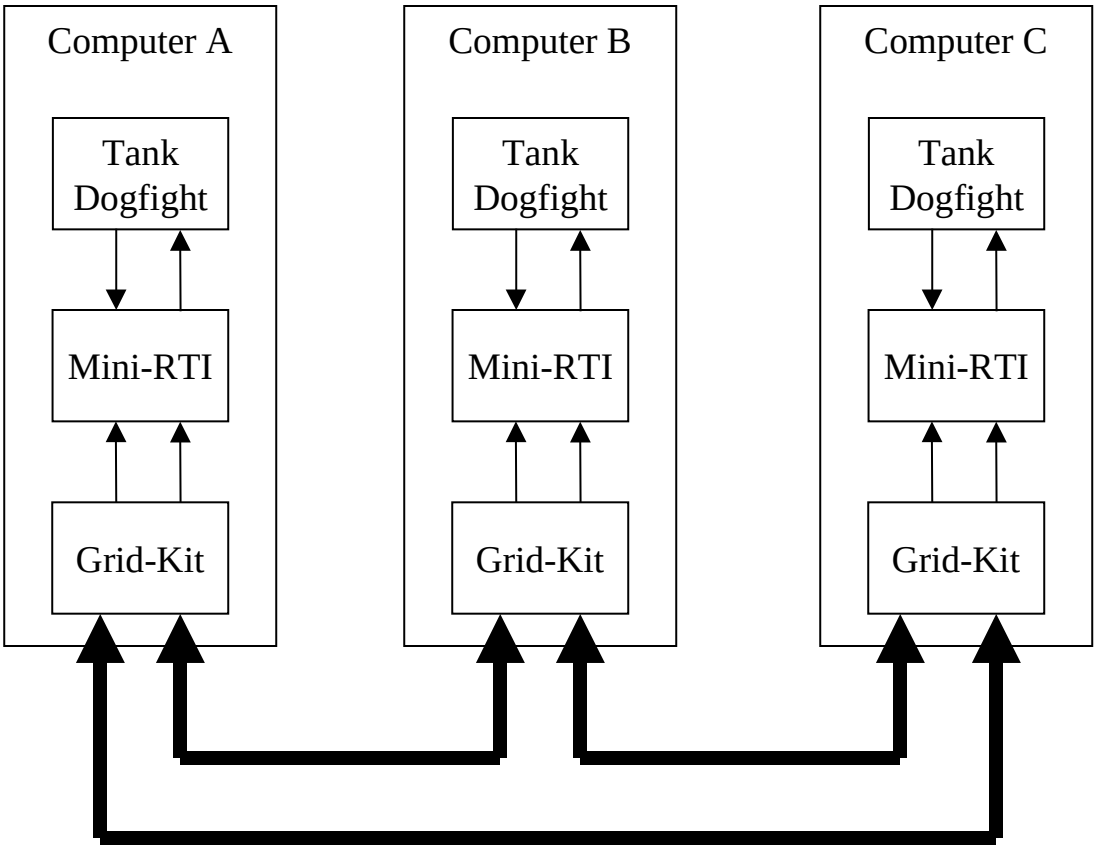

Figure 2. Peer-to-Peer Architecture

1.4 Thesis Organization

This thesis focuses on problems in the area of large-scale distributed simulation, specifically interactive military training simulations. Our main objective is to design and implement a new Data Distribution Management (DDM) mechanism. The purpose of DDM is to limit and control the volume of data exchanged during the execution of the simulation and reduce the processing requirement of the simulation hosts by relaying the events and state update information only to those simulation hosts that need them.

As discussed earlier in this chapter, the three components of our software system are the Tank Dogfight federate, the UNT-RTI, and the Grid-Kit. The Tank Dogfight federate can be viewed as an application that uses the Grid-Kit, and the UNT-RTI is the interface between the two. In Chapter 2, we describe the API for this interface using the High Level Architecture (HLA) specification.

In chapter 3, we discuss several RTI prototypes that have been developed by now. We also present a classification of these RTIs. Finally we present our UNT-RTI that we have developed. The purpose of the UNT-RTI is to test our Grid-Kit, which is the part of our system that implements our new method of Data Distribution Management (DDM).

In Chapter 4, we describe previous and related work in DDM. We classify the DDM approaches into two groups: region-based and grid-based methods. We discuss the strengths and weaknesses of these two strategies.

In Chapter 5, we describe our DDM scheme, which we refer to as a dynamic grid-based multicast group allocation, and we present an example to illustrate how our DDM scheme works.

In Chapter 6, performance measurements and an analysis of our DDM scheme are described. Extensive simulation experiments performed with the UNT-RTI, using real-world scenarios, are also presented. A comparative study of our DDM scheme with previous DDM schemes is also discussed in Chapter 6.

Finally, in Chapter 7, a summary of our thesis and a brief account of future extensions to our work are presented.

2. HIGH LEVEL ARCHITECTURE (HLA)

The U.S. Defense Modeling and Simulation Office (DMSO) developed the HLA in the late 90's to support reuse and interoperability among the many different types of simulations developed and maintained by the U.S. Department of Defense (DoD). The Institute of Electrical and Electronic Engineers (IEEE) is now in the process of approving the HLA as an open standard. It is hoped that once the HLA becomes and IEEE standard, the application of HLA will spread from the defense and military sectors to private corporations. Already several universities are doing research and development based on the HLA, and our work at the University of North Texas is one such example.

The High Level Architecture (HLA) represents a common framework within which compliant simulation components, called *federates*, can be interconnected to build federations of cooperating simulations. The purpose of the HLA is to provide federates that are distributed among several hosts a common framework that furnishes services needed by distributed simulations. This strategy provides interoperability, since diverse federates can communicate and participate in the same virtual environment. Reuse of software is also achieved, since federates can use the common services provided by the HLA without needing to implement these services themselves. Interoperability and reuse are the primary motivations behind the adoption of the HLA.

2.1 Federation Development

To better understand how the HLA can be used in the development of a simulation, let us examine a sample federation. Suppose we wish to simulate the movement of infantry forces. We wish to determine how long it will take a group of soldiers to march various routes, based on the equipment being carried, the terrain, and the weather conditions. Assume we have one program that simulates the ground troops, and calculates the speed at which they march, given these parameters. Let us also assume we have a second program that simulates terrain and environmental factors. Neither program was developed with the other one in mind. However, if both are HLA compliant, they should be able to interact with each other to produce a simulation of both troops and their surroundings. Allowing specialized simulators to join together to create more complex simulations is one of the primary purposes of HLA.

In this example, the infantry simulator and the weather simulator would be the two participating *federates*, and they would combine to form a *federation*. The "contract" between the federates which specifies what data they will exchange, and other information necessary for the success of the federation, is defined in the Federation Object Model (FOM). There is on FOM per federation. However, a federate can participate in one federation one month, and another federation the next. The Simulation Object Model (SOM) of that federate, which contains such information as what type of entities the federate can simulate, should remain constant, regardless of the federation in which it is participating. The information that should be included in the FOM and SOM and the format in which it should be presented, in specified in the HLA.

As evidenced by our discussion up to this point, the HLA is not only intended to influence software architecture, but also business practices. Simulator builders are encouraged to make their SOMs available, making known the capabilities of their systems. In this way, those users that are in need of simulations can inspect the SOMs of available simulators and determine if any one of them, or any subset of them that can be federated, might meet their needs.

Suppose the user chooses to federate existing simulators, based on the information contained in their SOMs and perhaps also consultation with the simulator owners, whose contact information is also included in the SOM. The next step would be for the user and the simulator owners to agree on the design of the federation. Specifically, all the elements required for a FOM should be resolved. These decisions would be captured in the FOM that would be used for their unique federation.

Reusing available software, instead of building new program from scratch, is a very desirable goal. Unfortunately, it is one that is difficult to attain without some higher level structure to which all the software must conform as well as guidelines as to how the collaboration process among human beings will occur. The HLA attempts to provide both a framework for software interoperability and a process for communication between the humans who build and use simulations.

2.2 HLA Rules

The HLA is comprised of three elements defined in three documents, as illustrated below. We will describe each of the three in turn, beginning with the HLA Rules.

1) A set of *HLA Rules* which establish guidelines and define relationships among federating compliant simulations

2) An *Interface Specification* which describes the way compliant simulations interact during operation

3) An *Object Model Template* Specification which specifies the form in which simulation elements are described.

The HLA Rules are one of the three elements of the HLA framework. They specify the principles characterizing the HLA. A federate must follows these rules in order to be judged HLA compliant. There are ten rules, and half of them address guidelines for federations, and the other half deal with rules governing federates. The rules are enumerated in Figure 3.

Rules for federations:

1) Federations shall have an HLA federate object model (FOM), documented in accordance with the HLA Object Model Template (OTM).
2) In a federation, all simulation-associated object instance representation shall be in the federate, not in the runtime infrastructure (RTI).
3) During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
4) During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification.
5) During a federation execution, an instance attribute shall be owned by at most one federate at any given time.

Rules for federates:

6) Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA OMT.
7) Federates shall be able to update and/or reflect any attributes and send and/or receive interactions, as specified in their SOMs.
8) Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOMs.
9) Federate shall be able to vary the conditions (e.g. thresholds) under which they provide updates of attributes, as specified in their SOMs.
10) Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

Figure 3. Summary of the HLA Rules

The complete HLA Rules documentation also provides further explanation of the rules and also gives the rational for why a particular rule has been included in the HLA. We will now highlight terminology that is used in the HLA Rules and in the HLA literature in general.

2.2.1 Basic Concepts in HLA

In the HLA Rules discussed in the previous section there are several terms which are common in object-oriented approaches but which have a specific meaning when uses in the context of the HLA. *Object* refers to an entity that is able to interact with entities being simulated by other federates. The types of objects being used by a federate is made publicly available to the other member of the federation by means of the FOM and SOM, which were mentioned earlier in this chapter.

Not all simulated entities will be objects, since some entities might be of interest only to one of the federates. The term *attribute* has a similar connotation, since all the attributes of an object are of interest to more than one federate. An object might have other properties associated with it that are used privately by a single federate, and such characteristics are not considered attributes by the HLA. It is possible for different federates to be managing attributes of the same object. *Ownership* in the HLA can occur on a per-attribute level, and is not restricted to a per-object basis. Only the federate that owns a particular attribute can modify the value of that attribute.

Now that we have define important terms associated with the HLA and used in the HLA Rules, we now focus our attention on the rule that relates most closely to data distribution management, which is Rule #3. This rules specifies how data exchange in the HLA is to occur.

2.2.2 Data Exchange in the HLA

The Data Distribution Management service is concerned with the exchange of data among federates. Let us now examine the HLA rule that deals with this important issue. Data exchange in the HLA is governed by the third HLA Rule, which states,

> *"During a federation execution, all exchange of FOM data among federates shall occur via the RTI."*

Recall that FOM data is information regarding the objects, and their attributes, that are being simulated by federates. Rule three implies that if federate A wishes to make the attribute values of one of its objects, call it OBJ, available to other federates, and if federate B wishes to receive information about OBJ, then the transfer of data between federates A and B must go through the RTI. RTI stands for *Run-Time Infrastructure,* which is the software that implements the HLA specifications and provides the HLA services to federates. We will discuss the RTI in detail, but first let us inspect the HLA Interface Specification that defines the interaction between the RTI and federates.

2.3 Interface Specification

The interface specification declares the methods, i.e. function calls, that federates must use to access HLA services. There are six groups of services defined in the HLA.

1) *Federation Management* – creation, dynamic control, modification, and deletion of a federation execution
2) *Declaration Management* – allows federates to declare their intention to generate or receive information

3) *Object Management* – registration, modification, and deletion of object instances and the sending and receipt of interactions
4) *Ownership Management* – transfer ownership of attributes among federates
5) *Time Management* – controlling the advancement of each federate along the federation time axis
6) *Data Distribution Management* – further refines the services provided by Declaration Management

2.3.1 Declaration Management vs. Data Distribution Management

Data Distribution Management (DDM) is the services with which we are most concerned. However, the listing of HLA services in the preceding section mentions Declaration Management, which is somewhat similar to DDM. We shall now discuss the similarities and differences between these two HLA services.

Declaration Management provides data relevance information in terms of object classes, which are defined in the FOM. Although these object classes are not necessarily implemented by software classes such as those provided by the C++ language, there is a similar concept of inheritance. For example, there may be an object class Airborne that has subclasses Rotary_Wing and Fixed_Wing. Fixed_Wing may have subclasses Prop and Jet, which may be specialized by classes representing particular models such as SR-71 or F-14. Using the Declaration Management service, a federate can request to receive information concerning objects of the Airborne class, which will include all classes that have Airborne as a superclass. A federate can also express interest at the subclass level, such as Jet, or F-14.

Using the Data Distribution Management service, a federate can not only specify the class of interest, but also the region of interest. The region is a subset of a space, known as the *Routing Space*, that is defined in the FOM. For example, suppose that the routing space had a dimension called Continent. A federate could express interest in Jets located in North America. Thus, Data Distribution Management allows more specificity in expressing data requirements than does Declaration Management. A federating might not need this added level of detail, and may elect to use only Declaration Management. However, in a large, complex federation with a high number of objects the ability to express interest both in terms of object class and region, as provided by the Data Distribution Management service, is a very useful feature.

2.3.2 Specification Excerpt

In order to give a concrete example of the HLA Interface Specification, Figure 4 presents an excerpt from the Federation Management services chapter. The service is *Create Federation Execution*. The description contains enough precision so that it may be easily understood, but does not express exactly *how* the service is to be implemented. The *Related Services* section lists other HLA services that are associated with the service being described and acts as a cross-reference to other parts of the document.

4.2 Create Federation Execution

The *Create Federation Execution* service shall create a new federation execution and add it to the set of supported federation executions. Each federation execution created by this service shall be independent of all other federation executions, and there shall be no inter-communication within the RTI between federation executions. The FED designator argument shall identify FED that is required for the federation execution to be created.

Supplied Arguments
- Federation execution name
- FED designator

Returned Arguments
- None

Pre-conditions
- The federation execution does not exist.

Post-conditions
- A federation execution exists with the given name that may be joined by the federates.

Exceptions
- The federation execution already exists.
- Could not locate FED information from supplied designator
- Invalid FED
- RTI internal error

Related Services
- *Destroy Federation Execution*

Figure 4. HLA Interface Specification Excerpt

In addition to this type of language independent description of all services, the Interface Specification also contains an annex for specific programming languages. An excerpt from Annex B, the C++ application programmer's interface (API), is given in Figure 5. The function *createFederationExecution* in Figure 5 is the same service that was described in Figure 4.

There is also an API for IDL, Ada, and Java in Annex A, C, and D respectively. As new programming languages are developed, additional annexes specifying the API for those languages can be added, potentially without altering the language independent descriptions giving in the main document of the Interface Specification.

```
///////////////////////////////////
// Federation Management Services //
///////////////////////////////////


// 4.2
void createFederationExecution (
  const char *executionName,
  const char *FED)
throw (
  FederationExecutinAlreadyExists,
  ConcurrentAccessAttempted,
  RTIinternalError);



// 4.3
void destroyFederationExecution (
  const char *executionName)
throw (
  FederationsCurrentlyJoined,
  FederationExecutionDoesNotExist,
  ConcurrentAccessAttemped,
  RTIinternalError);
```

Figure 5. HLA C++ Application Programmer's Interface

2.3.3 Implementation of the HLA

As evidenced by Figure 5, the HLA API can be liked to a header file that specifies the functions that need to be invoked to perform certain tasks. The software that implements these functions is the Run-Time Infrastructure (RTI). The RTI provides a set of services, as defined in the HLA, that allow federates to interact with one another in the execution of a federation. All federation communications must go through the RTI, as we noted earlier in this chapter when we discussed the HLA Rules. It is important to

understand that the HLA is NOT the RTI. We can refer to *the HLA* since there is only one Interface Specification and one API (per programming language). However, we may refer to *an RTI* since different RTIs have been developed that perform the functions specified by the HLA. When we refer to *the RTI* we are not discussing a particular RTI but are referring in a general way to characteristics or functionality that applies to all RTIs.

Continuing with our analogy of likening the HLA API to a header file, the RTI can be seen as the source code that carries out the functions declared in the header file. Just as different algorithms and programming constructs can be used to implement the same functions, there can be many RTIs that are compliant with the HLA. An RTI can be a separate process from the federates, or it can be linked to the federate executable. Parts of the infrastructure may be distributed to various hosts, but although it may be running on the same machine as a federate, it is both conceptually and concretely separate from and independent of the federate. A federation could theoretically be run using one RTI, and then be executed again using another RTI. The federates would probably need to be re-linked so that the HLA calls would invoke the other RTI, but they should not need to be recompiled. We shall devote a later chapter to discussing the RTI framework and the various RTIs that have been build to date.

2.3.4 Performance and HLA

There are clearly characteristics of the HLA that would suggest certain performance trends under certain condition. However, in order to measure the performance of the HLA in terms of memory, messages, running time, or other parameter

we must have a particular RTI in mind. The HLA defines a set of services and their API, but the expedience with which these services are performed is dependent on the particular RTI being used by a federation. With advances in computer science, including networking, operating systems, databases, programming languages, et cetera, new RTIs can be build which will take advantage of the latest hardware, software, and programming techniques. It is hoped that the HLA approach of specifying an API but allowing the implementation (i.e. the RTI) to be modified, and even recreated, as technology changes will enable the HLA to meet the needs of the simulation community for years to come.

2.3.5 Distributed Interactive Simulation (DIS) Protocol

Before the development of the HLA/RTI framework, another approach to promoting interoperability was used. We give a brief overview of this method for historical reasons, and also to highlight ways in which the HLA/RTI seeks to improve on lessons learned from DIS.

In the 80's and early 90's, the Distributed Interactive Simulation (DIS) communications protocol allowed diverse simulation systems to interact with each other. The Protocol Data Units (PDUs) which are to be exchanged among the simulated entities (or hosts) specify this protocol.

In order for two applications on different hosts in a DIS-compliant distributed simulation to interact, they must exchange PDU that are sent and received directly over the network. Here, we are referring to simulations that are strictly DIS-compliant and not those that use the DIS PDUs but have added additional software. This software may be

complex and needs to be known by the distributed applications, giving its characteristics similar to that of an infrastructure and not necessarily a pure-DIS compliant system.

It is important to differentiate between an infrastructure, such as the RTI, and a protocol like DIS. DIS is a set of rules and conventions, and not executable software like the RTI. As such, the influence of DIS is confined to the data-link layer of the distributed simulation system. There is no DIS protocol component that mediates between distributed applications and the physical network, whereas the RTI software acts as a "middleware" layer [16]. One of the advantages of the RTI strategy is the federate builders need not be concerned with data packets being sent over the network. All communication between federates is handled by the RTI. This division between the domain specific federates that simulate planes, soldiers, and other objects and the domain independent RTI that manages low-level communications is one of the motivations behind the adoption of the HLA.

2.4 Object Model Template (OMT)

We have discussed two of the elements of HLA, the HLA Rules and the HLA Interface Specification. We now turn to the third and final element, the Object Model Template. The Object Model Template (OMT) specifies how to document important information about simulations and federations. The OMT is used to develop SOMs and FOMs. As described earlier in this chapter, the SOM (Simulation Object Model) lists the data requirements, objects classes, and other information about a particular federate. The FOM (Federation Object Model) provides federation-wide information. Since the OMT

presents a standard format for the FOM and SOM, software tools can automate the creation and maintenance of these documents. Potential automation of the process is one of the benefits of the OMT. Another advantage is format consistency among federates and also among federations, promoting ease of understanding for federation builders and users. Recall that the HLA is not only intended as a technical specification, but also seeks to influence business practices by encouraging increased collaboration and greater understanding among all parties involved in the federation development process.

2.4.1 OMT Example

Table 1 shows an example of the *object class structure table* of the OMT, which can be used in both FOMs and SOMs. We have filled it in according to object class names that we used in an example earlier in this chapter. For conciseness we have omitted the letters *PS* that would usually appear in parenthesis following the class names. These letters refer to Publishable and Subscribable, two terms related to Data Distribution Management, which we shall discuss in detail in a later chapter.

The table shows the class names, and also the inheritance structure of the classes. If cell *i* is immediately to the left of cell *j*, then the class named in *j* is a subclass of the class named in *i*. For example, Surface_Ship and Submarine are subclasses of Maritime, and Destroyer, Cruiser, and Battleship are subclasses of Surface_Ship.

Object Class Structure Table			
Airborne	Rotary_Wing		
	Fixed_Wing	Prop	
		Jet	SR-71
			F-14
Ground_Based	Tank		
Maritime	Surface_Ship	Destroyer	
		Cruiser	
		Battleship	
	Submarine		

Table 1. Object Class Structure Table Example

2.5 Data distribution Management and non-HLA simulations

In this section, we briefly discuss why Data Distribution Management (DDM) is an important part of an RTI. We also present the conditions under which non-HLA simulations can also take advantage of DDM.

Data Distribution Management has now become a common, if not a mandatory function of an infrastructure supporting large scale distributed simulations. DDM is being used by several simulation systems that are HLA-compliant, as well as other environments. However, not all distributed simulations are equally able to take advantage of the optimization provide by the DDM. Data Distribution Management can only be used in systems when the data representing the virtual world is distributed

according to the shared world environment, which is used by the HLA, as opposed to the replicated world, which was used by DIS. We now compare the replicated world and shared world strategies.

2.5.1 Replicated World

Simulations that use protocols for inter-hosts communication, such as DIS-compliant systems, do not have any middleware that can store or coordinate the state of the virtual world. The protocol-based method of implementing a federation of diverse simulators implies the use of a replicated virtual world. Each simulator must keep track of the state of the virtual world in which the federation operates, meaning that it may be replicated on many of the hosts that are part of the federation. Clearly this is inefficient in terms of host memory usage as well as network traffic, since every update to the virtual world must be transmitted to all hosts.

DDM is not applicable in a replicated world environment, since by definition all hosts need all data associated with the state of the world. Thus, the paradigm of replicated worlds disallows doing any data filtering. Most DIS-compliant simulations fit into the replicated world model. However, it would be more accurate to categorize them as *stateless* systems, i.e., they have no notion of a shared virtual world. DIS uses IP/UDP broadcasts to send to every host every state change that occurs within the virtual world. The simulators must then construct their own world based upon the messages they have received. In addition, DDM cannot be used when there is no middleware layer to mediate between the simulations and the network, which is true for DIS-compliant systems. Some DIS-compliant systems have been modified, and so no longer use the

DIS broadcast protocol. Instead a middleware has been added to perform some of the DDM services. The DoD is encouraging all DIS simulations to be HLA-compliant, and thus to use the shared world paradigm in order to benefit from the DDM services.

2.5.2 Shared World

The observation that all hosts do not need all data lead to the conclusion that the replicated world paradigm used by DIS is generally not very efficient. Fortunately, HLA federations that use an efficient RTI can avoid it. The RTI can serve to store the data associated with the state of the world among the hosts, and thereby simulating a shared memory architecture using the shared world paradigm, as illustrated in Figure 6. One of the main features of the RTI is to provide an effective DDM service and ensure that each host gets, in a timely manner, the data that it needs.

Figure 6. Shared World

This feature shall help to reduce the use of memory and the processing power, by reducing the total amount of data that needs to be received, evaluated, and stored. However, these gains can be easily negated by the additional network load of the synchronization messages transmitted, the processing time incurred by the RTI while performing the necessary computations, and the potential bottleneck while requiring the data that will be managed by the RTI. These overheads are the result of the additional

coordination and organization required by having the data distributed among the various hosts, and ensuring that the hosts shall receive only that the data they are interested in.

DDM mechanisms seek to meet the data requirements of hosts while minimizing its associated overhead. Inefficient DDM schemes can have a devastating effect on the performance and the scalability of any distributed simulations. Thus, DDM is necessary and it is a critical part of the RTI, and of any infrastructure that shares the data relevant to the state of the virtual world.

2.6 Summary

In this chapter, we have discussed some basic concepts and the three elements of the High Level Architecture (HLA). We also noted alternatives to the HLA approach, such as the Distributed Interactive Simulation (DIS) protocol. In the next chapter we focus on the software that implements the HLA specification, namely the Run-Time Infrastructure (RTI).

3. RUNTIME INFRASTRUCTURES

Runtime Infrastructure (RTI) is software that implements the services defined in the High Level Architecture (HLA). The RTI allows diverse simulators, called *federates*, to interact with each other and participate in the same virtual environment. As discussed in the previous chapter, these services fall into the six general categories of federation management, object management, ownership management, declaration management, time management, and data distribution management. These are common services that most distributed simulations need to use in order to operate accurately and effectively.

One of the advantages of the HLA is the *separation of concerns*. By placing the responsibility of implementing the six HLA services on the RTI and freeing federates from that obligation, federate builders can focus on how to simulate the domain-specific objects they wish to represent, such as planes or tanks. The developers of the RTI, in turn, can concentrate on how to implement the HLA services and do not need to be concerned with application-specific information. Although working independently of each other, both the developers of federates and RTIs must adhere to the HLA specification, and the API specific to the programming language being used, in order for their software to cooperate correctly.

The HLA specifies the API that federates use to access the RTI services. It is important to keep in mind, however, that the HLA does not define certain algorithms, data structures, or network protocols that the RTI must use. RTI builders have significant freedom and a myriad of options in how they chose to carry out the HLA services.

In the previous chapter we discussed the HLA, and now we turn our attention to RTIs that have been developed. Recall that the HLA is a standard developed by the U.S. Defense Modeling and Simulation Office (DMSO) with the purpose of promoting interoperability and reuse among simulations. Although there have been RTIs build under the direction of DMSO, other groups have also created their own RTIs.

In this chapter we describe the RTIs that have been constructed under the sponsorship of DMSO, namely the RTI Prototype and RTI 1.3, as well as the RTI-NG that is currently under development. We also characterize the RTIs that have been build commercially by MAK Technologies and RAM Labs. In addition, we shall investigate components that, while not complete RTIs, are being used as frameworks in which to research the most efficient way to perform specific HLA services. Examples of such systems are the RTI-Kit developed at the Georgia Institute of Technology and the LightWeight RTI built at George Mason University.

Finally, we shall describe the Grid-Kit and UNT-RTI that we have created at the University of North Texas. The Grid-Kit module works within the framework of the RTI-Kit and can be used to perform the Data Distribution Management service. The UNT-RTI is an interface between the Grid-Kit and the federate and enables us to perform simulation experiments to assess the performance of the Grid-Kit, which we describe in a later chapter.

3.1 DMSO RTIs

3.1.1 RTI Prototype

The RTI Prototype was government-developed in the mid-90's at Massachusetts Institute of Technology (MIT) Lincoln Laboratories [8]. The HLA specification was still under development and undergoing revisions during this time period, so the creators of the RTI Prototype did not implement all of the HLA services (for example, Time Management was omitted) or follow the specification to the letter. Instead, their implementation was designed as a proof of concept for HLA, with the intent of also providing feedback that might impact the development of the HLA itself. In this section we summarize the technical approach taken in the RTI Prototype, including the Data Distribution Management strategy, and then we describe the experimental results for the RTI Prototype.

The API for the RTI is specified in the following in languages: IDL, C++, Java, and Ada. IDL stands for Interface Definition Language , a part of the CORBA specification. The RTI Prototype uses the IDL API specification, and was developed using C++ and the Iona ORBIX implementation of CORBA. The approach to Data Distribution Management (DDM) used in the RTI Prototype is know as the Fixed Grid-Based Approach, which we discuss in detail in our chapter concerning various approaches to DDM.

Before the RTI Prototype was tested in the Synthetic Theater of War (STOW) federation, laboratory experiments were conducted [6]. The experimental results

available deal with only two federates. These experiments seem to have been designed to verify that the RTI Prototype was indeed functioning properly, and not to show how it performs when each federate is simulating many objects which are moving around in the battlespace, as no such scenario was performed. When the STOW exercise was conducted, a data logger captured statistics, such as the total number of packets sent over the network. These statistics seem to be confined to the data-link level of the simulation, where as the HLA deals with a higher level of abstraction. Therefore, this data unfortunately does not give direct insight into the performance of the RTI Prototype or the Data Distribution Management system.

3.1.2 RTI 1.3

The same group at Lincoln Labs that designed the RTI Prototype developed the RTI 1.3 [30]. The RTI 1.3 is the successor of the RTI Prototype and is so named because it implements version 1.3 of the HLA Specification. One of the main changes between the RTI Prototype and the RTI 1.3 was the Data Distribution Management strategy that was used. The RTI Prototype used a grid-based approach, whereas the RTI 1.3 employs a region-based method. The region-based method is described in detail in our chapter concerning previous DDM work, where we note that a distinguishing feature of this approach to DDM is the use of a single database to store information regarding the regions of interest declared by all federates in the federation. The database for regions, subscriptions, and publications used by the RTI 1.3 is called the Information Manager (IM). Unfortunately, no experimental results have been published concerning the performance of the RTI 1.3.

3.1.3 RTI 1.3NG

DMSO sponsored the RTI 1.3NG and awarded the software development contract to Science Applications International Corporation (SAIC) based on competitive industry designs, unlike the contract for the RTI Prototype and RTI 1.3, which was not based on open competition among companies specializing in defense contracting [10]. Like the RTI 1.3, the RTI 1.3NG also supports the HLA Specification 1.3. However, the RTI 1.3NG is intended to be a full implementation of all HLA services and will supersede the RTI 1.3, which is no longer being supported by DMSO [24] . As yet, no literature has been published regarding the design, implementation, or performance of the RTI 1.3NG, which is still under development.

3.2 Other RTIs

We have described the RTIs that have been build under the sponsorship of DMSO. Now we shall discuss several other RTIs that have been built by corporations, most of which specialize in defenses applications.

3.2.1 MAK RTI

MAK Technologies, which is based in Cambridge, Massachusetts, and is a leading provider of simulation networking software, developed the MAK RTI. It supports the HLA Specification 1.3, and is link-compatible with DMSO's RTI 1.3. Like the other RTIs that we have discussed, the MAK RTI does not implement all HLA

services. The MAK RTI can be downloaded at no charge, and runs on the Windows 95/98/NT, Solaris, IRIX, and Red Hat Linux platforms.

3.2.2 HPC-RTI

RAM Labs, based in Solana Beach, California, is developing an RTI designed for use in high performance computing (HPC) environments. The RTI-HPC is integrated with the Synchronous Parallel Emulation Environment for Discrete Event Simulation (SPEEDES) [26]. SPEEDES is a government-owned software system, managed by RAM Labs, and licensed by NASA. SPEEDES was used in the early 90's to model global ballistic missile defense applications on parallel and distributed supercomputers [27]. SPEEDES currently supports various large-scale distributed simulation projects, under the sponsorship of the Department of Defense, such as Wargame 2000, Joint Simulation System (JSIMS), and Extended Air Defense Test Bed (EADTB) [25]. SPEEDES is implemented in C++, and supported operating systems are IRIX, HPUX, Solaris, Linux, and Windows NT.

With the HLA gaining popularity in the defense community, SPEEDES is being modified and augmented to serve as an RTI that is compatible with the HLA specifications, and the result will be the RTI-HPC. The RTI-HPC is the first attempt to transform a pre-existing simulation engine into an RTI. There is another element of the RTI-HPC that differentiates it from the other RTIs, and that is its support of time management.

The RTI-HPC will differ from the other RTIs we have discussed because it will provide time-management across all six HLA services. The HLA does not require the

Declaration Management, Data Distribution Management, or Ownership Management services to be time-managed. However, such capabilities would be extremely useful for a federation that wanted to enforce casual ordering of events, for the purpose of repeatability or other reasons. (The mechanism by which SPEEDES performs Time-Management is intricate and is beyond the scope of this thesis.) In the remainder of this section we will discuss how SPEEDES performs Data Distribution Management.

The SPEEDES Data Distribution Management mechanism follows a grid-based approach [26]. Interest regions are mapped to grid cells, which are represented by entities called Hierarchical Grid (HiGrids) where region-overlap computations are performed. The HiGrids are distributed among the participating nodes. When an overlap is detected, HiGrids tell the publisher which subscribers are interested in the publisher's attributes. The publisher then sends the subscriber its proxy, which represents the state of an entity. The proxy is updated whenever the publisher's attributes change, thus ensuring that the subscriber can assess the publisher's current attributed values.

Experimental results, taken on a 96 processor Origin 2000 using shared memory communications, have verified the correctness and efficiency of the SPEEDES DDM mechanism. A scenario modeled 1,000,000 entities, each with a sensor range of 100 kilometers. Each entity moved in great-circle trajectories around the earth in a random fashion, with randomly generated velocities between 0 and 0.333 mach. Results showed a nearly perfect speedup, and a memory increase of only 10%, when going from 32 to 96 processors. This indicates that the message passing overheads and memory usage scales appropriately.

The experiment described above that evaluated the performance of SPEEDES DDM has yielded impressive results. However, The SPEEDES method of performing data distribution management was developed prior to the advent of the HLA specification, so some changes will need to be made in the transition to the HPC-RTI. The concept of a proxy is not fully compliant with the HLA specification. In addition, SPEEDES allows subscriptions and publications on a per-object basis, whereas the HLA only allows interests to be declared on a per-federate basis. The use of proxies, and the per-object interest expressions, are features that may need to be modified as SPEEDES is converted to the RTI-HPC. It remains to be seen how closely the RTI-HPC method of supporting the HLA Data Distribution Management service resembles the current SPEEDS approach to DDM.

We have summarized the DMSO-sponsored and commercial RTIs that have been developed. We now turn our attention to the RTIs that are being used in the academic sector for the purposes of research and experimentation. Since research groups at universities generally have a much smaller funding with which to work on HLA-related projects, as compared to companies or government contractors, the academic RTIs tend not to be fully functioning RTIs that are HLA-compliant in all respects. Instead, they usually focus on implementing a specific part of the HLA specification, ordinarily those services of interest to the principle researchers. The smaller scope of these RTIs might be seen as a drawback, since it would not be practical to use these systems to support a large-scale exercise that required all of the HLA services. However, the narrower focus of these RTIs can also be viewed as an advantage since this approach allows their

developers to isolate specific aspects of the HLA services and pay careful attention to the algorithms, data structures, and programming techniques that are used in their implementation. Academic researchers also have great freedom to investigate new approaches and to pioneer new ways to build RTIs.

3.2.3 GMU Light-Weight RTI

The light-weight RTI developed at George Mason University (GMU) focuses on Declaration Management and the Data Distribution Management services [20]. Time Management and Ownership Management are not implemented, since these services were not the primary objective of the project. As a result, the light-weight RTI is best suited to real time simulations by federation of small to medium size. A useful feature of this RTI is that it can be interfaced to DIS simulations using a DIS to HLA translator developed at GMU [17]. The motivation behind its construction was to understand the HLA and to bring the earlier work done by those researchers into compatibility with the HLA. We shall now briefly describe the previous works out of which the light-weight RTI grew.

The light-weight RTI uses elements of the Dual Mode Multicast scheme and the Selectively Reliable Transmission Protocol. Dual Mode Multicast (DMMC) is a method of Data Distribution Management that was developed for use by systems adhering to the Distributed Interactive Simulation (DIS) protocol. DMMC uses a multi-level grid-based filtering scheme. An exercise-wide multicast group is used on the wide area network (WAN) and a fixed grid-based approach is used to determine multicast groups at level of the local area network (LAN) [12].

Selectively Reliable Transmission Protocol (SRTP) is designed for applications, such as DIS and HLA, that need reliable multicast communications. SRTP runs in user space and forms a sublayer between an application and the Internet protocol stack of the operating system. SRTP operates in three modes: best effort multicast reliable multicast, reliable multicast, and lightweight reliable transaction-oriented unicast. The reliable multicast uses negative acknowledgement with NAK suppression mechanisms to avoid congestion at the sender [15].

The major flaw in the light-weight RTI, according to its creators, is the poor runtime performance that is partly due to the use of SRTP, which is slower than UDP. Improving the performance of the light-weight RTI is the primary goal of the future work on this project.

3.2.4 RTI-Kit

The RTI-Kit, developed at Georgia Tech [14], contains a set of libraries designed to support development of Run-Time Infrastructures (RTIs) for parallel and distributed simulation systems, especially federated simulation systems running on high performance computer platforms. It is envisioned that these libraries will be embedded into existing RTIs, e.g., to add new functionality or to enhance the performance of the federated simulated systems while exploiting the capabilities of the high performance interconnection architecture. Alternatively, the libraries can also be used in the development of new RTIs. The major components of the RTI-Kit are:

- **FM** – communication layer software, API based on the Illinois Fast Messages
- **MCAST** –multicast group communication services

- **TM-Kit** – time management services, calculates lower bound on time stamp
- **MB-Lib** – allocating and releasing message buffers

None of the above modules handle Data Distribution Management. However, a **DDM-Kit** is current under development. The DDM-Kit will use the region-based method of Data Distribution Management [14].

As noted earlier in this section, the RTI-Kit is not intended to be a complete RTI, but instead is a set of building blocks for RTI research and development. As a proof of concept, an RTI based on the RTI-Kit has been constructed and used in academic research. It is known as DRTI[1] and its primary purpose is to exercise the functions provided by the RTI-Kit. It does not support all aspects of the HLA specification, and does not provide any Data Distribution Management or Ownership Management services.

We now summarize the salient features of the RTIs that we have discussed in this chapter. Table 2 presents a comparative study of the RTI software that is currently available. The *Name* of the RTI is chosen by its creators, and unfortunately is sometimes not very descriptive. The *Developers* column portrays the organization that is primarily responsible for the construction of the RTI. For the universities this identification is straightforward, but keep in mind that for the non-academic organization there may be several companies, in addition to the given prime contractor, that participated in RTI development.

We have also categorized these RTIs by their *Purpose*. We have labeled an RTI *R&D* if its developers intended it to be used for research purposes or as a starting point

for more extensive development. We identify an RTI as *Product* if its creators intended it to be used for large-scale distributed simulation projects managed by a third party. We have also classified these RTIs by their approach to Data Distribution Management. For some of these systems, the information has not been made published, hence the value of *<unknown>* in the *DDM Method* column.

Name	**Developers**	**Purpose**	**DDM Method**
RTI Prototype	MIT Lincoln Labs	R&D	Region
RTI 1.3	MIT Lincoln Labs	Product	Grid
RTI 1.3NG	SAIC	Product	<unknown>
MAK RTI	MAK Labs	Product	<unknown>
HPC-RTI	RAM Labs	Product	Grid
Light-weight RTI	George Mason Univ.	R&D	Grid
DRTI (w/ RTI-Kit)	Georgia Tech. Univ.	R&D	Region[2]

Table 2. Comparison of RTIs

We have now discussed and classified the RTIs that have been developed both by universities and private companies. We shall now turn our attention to the RTI that we have developed at the University of North Texas.

[1] Debbie's RTI (DRTI) is named after the person who lead the project
[2] Refers to the design of the DDM-Kit. Implementation has not yet been completed.

3.3 UNT-RTI

Just as the primary purpose of the DRTI, which we discussed earlier in this chapter, is to exercise the RTI-Kit, the intent of our UNT-RTI is to serve as an interface to the Grid-Kit [5]. For this reason, only the Data Distribution Management services are supported. In this section we will describe the relationship between our UNT-RTI and the RTI-Kit. In a later chapter we shall discuss the UNT-RTI and the Grid-Kit in more detail.

We have implemented a component called Grid-Kit that integrates with the RTI-Kit and performs DDM using our dynamic grid-based scheme. The various components of the RTI-Kit can be used together, or individually, depending on the needs of the user. In this context, the *user* would be an RTI builder or researcher. The RTI-Kit is freely available for download, including the source code. We found the source code to be reasonable well commented, and the accompanying documentation, although inconsistent in some instances, was overall quite helpful.

When we began development on the UNT-RTI, there was no RTI-Kit component that performed Data Distribution Management. Such a component, called the DDM-Kit, is currently under development. However, the design of the DDM-Kit uses the region-based approach to data distribution management, which has been implemented in the RTI Prototype. Our UNT-RTI uses the Grid-Kit that we have developed, which uses our new method of DDM called dynamic grid-based data distribution management. In addition to used the Grid-Kit, the UNT-RTI also uses certain components of the RTI-Kit. Of the

several RTI-Kit components that have been completed, the one that is more important to our work is the MCAST library, which we shall now discuss.

3.3.1 Multicast Group Communications

The MCAST-Lib component of the RTI-Kit provides our UNT-RTI with group communications services using the multicast paradigm. Currently the MCAST-Lib implements multicasting in software, meaning that the network does not need to multicast-enabled in order to use it. MCAST-Lib achieves communication over the network as follows. It uses another component of the RTI-Kit, the FM-Lib, which is communication layer software, API based on the Illinois Fast Messages (FM). FM-Lib can be compiled using various transport protocols, and we have used the TCP/IP version for our work. Note that to migrate to another communications mechanism, such as shared memory messaging, only the FM-Lib library would need to be modified not MCAST-Lib nor our UNT-RTI.

Analogously, optimizations made to the MCAST-Lib will not effect the UNT-RTI, as long its API remains unchanged. If additional functionality is added to the MCAST-Lib, our UNT-RTI can be modified accordingly to take advantage of these properties. Foremost among the enhancements to the MCAST-Lib that would be beneficial to our work is the implementation of a function to *delete* multicast groups. There is a method to create and initialize a group, but no way to indicate that a particular group is no longer being used.

To circumvent the problem of not being able to *free* multicast groups once they are allocated, the UNT-RTI *recycles* groups that are not needed. Once a group is no

longer in use, it is placed on a *free list*. Before creating a group, the UNT-RTI checks the free list. If the free list is not empty, a group is removed from the free list and reused. A new group is only created and initialized if the free list is empty.

In addition to using MCAST-Lib to create multicast groups, the UNT-RTI also uses it to join and leave groups. It is the UNT-RTI, by way of the Grid-Kit, that determines when federates should join and leave groups. However, the MCAST-Lib manages the actual membership lists for each group.

4. DATA DISTRIBUTION MANAGEMENT

Data Distribution Management (DDM) is an interesting and fundamentally challenging problem for the parallel and distributed simulation community. *Sequential* simulations can permit direct access between simulated entities since all entities live on one machine and progress consistently in time. On the other hand, a *distributed* simulation environment must closely regulate data access between simulated entities. Indeed, objects can be at different logical times (even within one processor), and it is difficult to reference remote objects. The aim of DDM is to limit and control the volume of the data exchanged during the simulation, and reduce the processing requirements of simulation hosts by relaying events and state information only to those applications that require them [18, 19, 27].

The fundamental observation leading to this strategy is that real-world objects might be interested in only a fraction of the objects surrounding them. For example, as illustrated in Figure 7, a spy plane is primarily interested only in other planes that are close to it. It is probably not concerned with the position of any other planes, some of which might be thousands of miles away. Systems without DDM, however, do not take into account this real-world phenomenon of limited interest. The host simulating the spy plane cannot elect to receive only information about certain objects, namely the ones that are within the radar range of the spy plane, which in this example are the planes in Squadron A. Instead, it gets an update message anytime any plane in Squadron A or

Squadron B, or any other aircraft, changes its position. All hosts will receive these state updates, just in case one of them should need it.

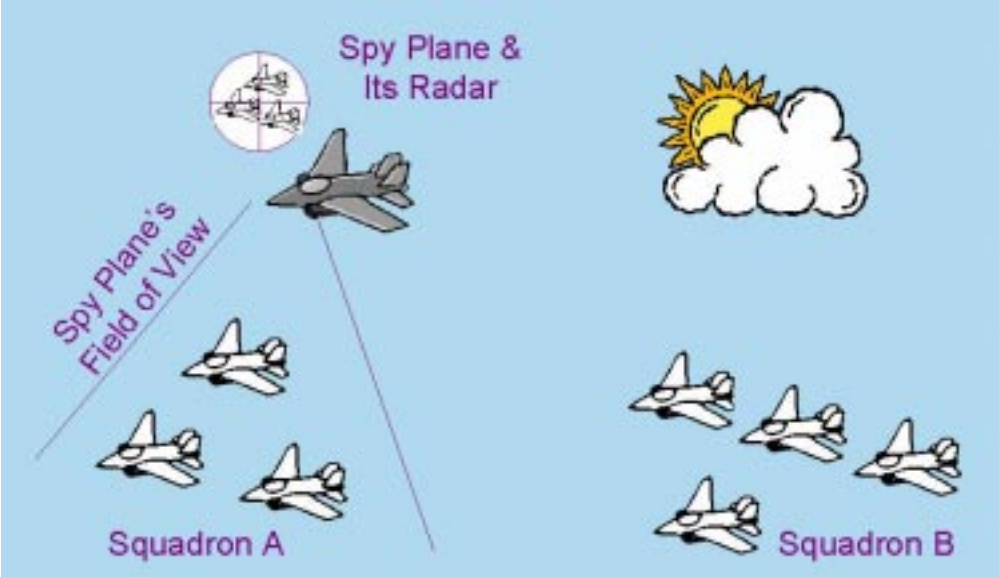

Figure 7. Domain Region of Interest

The costs of transmitting this irrelevant data can be very high. If a host receives a large amount of transmissions containing data that is irrelevant, the host will waste a significant amount of time and processing resources while receiving and reading it. Furthermore, sending data to a host that is not interested in it puts an unnecessary burden on the sending host and on the network.

In this paper, we propose a new DDM scheme, which we refer to as a Dynamic Grid-Based Approach. Our DDM *hybrid* approach is based on a combination of a Fixed Grid-Based method, known for its low overhead and ease of implementation, and a Region-Based strategy, which uses fewer multicast groups than the Fixed Grid-Based method.

4.1 Overview of DDM

In this section, we shall introduce the terminology that is commonly used in the DDM literature, and which will be fundamental to the discussion in the remainder of this paper [4]. In the course of our discussion, we will be using the same example of the spy plane and squadron planes used in the previous section.

4.1.1 Routing Space

A *routing space* is a multi-dimensional coordinate system [7]. It represents the virtual world in which the simulation occurs. In our example, we are using a two-dimensional routing space, one dimension being "up-down", the other "left-right," sine there is do "depth" to our picture. We have chosen to use a 2-D example first because it is the least complex case that fully illustrates DDM, and secondly because of the practical consideration of the difficulty of depicting a three-dimensional scenario on a two dimensional medium (i.e. this paper). The reader should keep in mind that practically all simulations use routing spaces that are at least three dimensions.

A routing space with at least three dimensions is almost obligatory when objects are able to both detect, and be detected by, other objects. In our two dimensional example, only the spy plane has a radar that detects the squadron planes. However, if we consider a more realistic scenario in which the squadron planes are also radar-enabled, the Squadron A planes will detect the spy plane. Note however that the planes in Squadron A will also detect each other, as will those in Squadron B. In most cases, those planes in Squadron A will not be interested in inter-squadron detections, and they will

only be concerned with detections of rival aircraft. If we assume that the squadron planes are on a different team than the spy plane, then, in order to properly simulate detections, the routing space would need a third dimension representing the *team* or *alliance* of which objects are members. In such a 3-D routing space, even if all the planes were radar enabled, detections of friendly aircraft could be distinguished from detection of unfriendly planes, and such discernment would be a key part of the simulation.

The third dimension described above, which represents the team to which the entity belongs, is an example of a dimension not based on geographical location. Other examples are *velocity* and *firepower*. A 5-dimentional routing space comprised of all these dimensions would allow detection of entities that were on a specific team, travelling in a certain area at a particular rate of speed, and carrying a specified compliment of weapons. Choosing how many dimensions a routing space will have, and what they will represent, is an important decision that must be made when designing a simulation that will use DDM.

4.1.2 Publication and Subscription Regions

A *region* is a subset of the routing space [7]. It is specified by a set of extents, one per routing space dimension. An *extent* is an ordered pair denoting the maximum and minimum ranges of a region along a particular dimension. A *subscription region*, or simply *subscription*, is an abstraction that refers to the set of world data in which a simulated entity (object or federate) is interested. A *publication region*, or *publication*, is an analogous construct that refers to the set of data that the simulated entity is making available to the world. A more general term, *interest region*, is sometimes used to refer

to both publications and subscriptions regions. In general, publication regions tend to be very small, sometimes even a single point, whereas subscription regions can be small or very large, depending for example on the radar capabilities of a simulated entity.

In our example, the spy plane's subscription region would be the area within its radar range. Simulated detections will occur whenever the publication region of a simulated entity, in this case the Squadron A planes, overlaps with the subscription region of the spy plane. A publication region represents where an entity can "be seen" and a subscribing region represents what it can "see," illustrating that a fundamental objective of DDM is determining "who can see whom."

A *subscriber* (or *publisher*) is the entity that has expressed an interest (or availability) and has generated the subscription (or publication). Publishers are the senders or producers of data, and subscribers are receivers or consumers. A publication region is also knows as an *update region*, since the publisher may need to send updates of its state information to its subscribers.

4.1.3 Intersection Region

The term *intersection region* refers to an area where subscription and publication regions overlap. Regions *overlap* if and only if the corresponding extent sets overlap [7]. If a given subscription and publication do not overlap, we say that there is no intersection (since there is no concept of an empty or null region). If an intersection region exists between a publication and a subscription, then data exchange occurs between the publisher and subscriber.

In our example, there is an intersection region where the publication of Squadron A overlaps with the subscription of the spy plane. Therefore, the spy plane receives state information that is sent by the Squadron A planes. Identifying the intersection and the subsequent exchange of state data is not a trivial process, since the entities involved may be simulated on different hosts. Let us assume the entities are distributed between three hosts[3] as follows: the spy plane is simulated on Node X, each of the Squadron A planes are on Node Y, and each of the Squadron B planes are simulated on Node Z. It is not immediately obvious how the spy plane on Node X knows that there is an intersection between its publication and a subscription on Node Y, and visa versa. As a consequence, there are two fundamental problems DDM must solve: (1) *Precisely how is the intersection region identified*, and (2) *how does the data exchange between publisher and subscriber take place?*

Since the mid 1990's, a considerable number of research projects on data distribution management in distributed simulation systems have been carried out in the literature due to the performance gains provided by DDM. We now examine this previous work. The DDM techniques can be classified into two groups: *region-based* [14, 28, 29, 31] and *grid-based* [1, 3, 7, 16, 20, 18, 19, 22]. They differ in the manner in which they determine intersections of publication and subscription regions.

[3] We use the terms *host* and *node* interchangeably

4.2 Region-Based DDM

The region-based approach, which was implemented in the RTI 1.3, compares the subscription and publication regions directly in order to find which ones overlap [30]. This process is known as matching. As is well known, in the worst case, matching requires every subscription to be compared with every publication, leading to O(N2) scaling characteristics. We shall now describe the two overheads associated with matching, which are computation and communication costs.

The computational overhead occurs because matching can happen frequently. Whenever a publication region is modified, it must be matched with all of the subscription regions. Publication region modification can be a very common occurrence. This is due to the fact that publishing objects, such as planes or tanks, are usually moving entities that repeatedly change their positions. Similarly, each time a subscription region is created or modified, this region must be matched to all of the publication regions on all federates. Subscription region modifications would be infrequent in the case of stationary sensing objects, such as ground-based radar, but could be numerous for any moving object equipped with a sensor.

The second expense associated with matching is the overhead of the inter-host communication needed to compare regions of different federates. Each federate creates and modifies its publication and subscription regions locally on its own host, but the matching must be preformed among regions federation-wide. The resolution is for the DDM system, which we will assume is a part of the High Level Architecture (HLA) Run-Time Infrastructure (RTI) [11], to have a federation-wide DDM_Coordinator, to which

all federates send their interest information. The DDM_Coordinator collects the interest information in a database, performs the matching among publications and subscriptions, and communicates the results back to the federates by triggering them to join or leave multicast groups, as explained later in this section. The use of intelligent mobile agents for collecting and managing interest information has recently been proposed in [28]. This is a promising technique for eliminating the centralized DDM_Coordinator from the region-based method.

The use of a centralized coordinator and database has the potential to have a crippling effect on the performance and scalability of the DDM system. This unfortunately is the major drawback of the region-based approach.

4.2.1 Multicast Group Communication

The communication among hosts and a DDM_Coordinator relates to the communication between federates and the RTI or among RTI components on different hosts. The purpose of this interaction is to allow the RTI's DDM mechanism to identify intersections. Once intersections are determined, a second type of communication must be enabled, and that is the sending of data from publishers to subscribers. This second kind of inter-host communication is generally performed using multicast technology.

In this paper, the term "*multicast*" refers to the concept of single-source, multiple-destination transmission. We believe that this type of communication is fundamental to all DDM implementations, and it can be implemented by various protocols, such as IP multicast, and TCP/IP sockets, just to mention a few. Systems that do not use DDM paradigm essentially use broadcast-like protocols, such as the DIS protocol [23]. In this

case, producers send data to all hosts, regardless of which hosts are consuming and/or interested in that particular data.

Let us look at how the region-based scheme uses the information gleaned from matching to assign hosts to multicast groups and facilitate data transfer. Suppose that the matching operation has determined that a publication made by federate *F* to region *p* is part of an intersection. Then, region *p* is assigned to a multicast group *mg-p.* Federate *F* is triggered to join *mg-p* and to begin sending data on that group. Subscriptions that intersect with *p* are also assigned to multicast group *mg-p* and their federates are triggered to join *mg-p* and thereby receive any data sent on that group. Thus, transfer of data has been facilitated between federate *F* and all other federates interested in the data *F* is publishing in region *p*. After additional matching has been performed, due to the addition, modification, or deletion of regions, suppose region *p* is no longer part of an intersection. Then, all federates that joined *mg-p*, including federate *F*, are triggered to leave the group, terminating the data exchange between publisher and subscriber(s).

4.3 Grid-Based DDM

The motivation behind the grid-based approach is to reduce the computational and communication overhead of the region-based method. While the region-based approach necessitates matching between all publications and all subscriptions, the grid-based technique avoids these extensive and costly operations. Instead of explicitly comparing publication and subscription regions, an RTI using a grid-based style maps each interest region onto a multi-dimensional grid, which represents the routing space, as shown in

Figure 8. (The grid will generally have the same number of dimensions as the routing space.) The subscription of the spy plane is mapped to Cells 1 and 3. The publication of each plane in Squadron A is mapped to Cell 3, similarly for the Squadron B planes and Cell 4. Cells that are part of both a publication and subscription represent the intersection, in our case Cell 3.

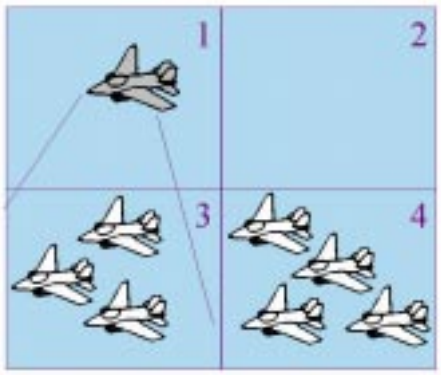

Figure 8. Grid Overlaid

Note that in Figure 8, only the subscription region overlaps multiple cells, while the publication regions are confined to one cell each. We believe that this is the most common case for military simulations, since publications regions are generally small and subscription regions are usually much larger, as noted in the previous discussion of interest regions. In the most general case multi-cell publications are possible. However, simulation designers could enforce a constraint requiring publications to be confined to one cell, and we call these point publications because they are limited to a single point or small region.

The process of overlaying a gird on the terrain can be performed by an RTI component on each host. Each one performs the region-to-cells mapping independently of the other hosts since the number and size of the grid cells are determined before the

simulations begins and remain constant throughout the run. A centralized DDM_Coordinator is not needed, making the grid-based method relatively scalable, which is its primary advantage.

The major drawback of the grid-based method is that its matching accuracy tends to be lower, as compared to the region-based method. The mapping of the interest regions to the grid cells may not be exact. Indeed, region boundaries might not fall precisely on cell boundaries, in which case the area covered by the cells representing the interest region will be slightly larger than the region itself. This may cause superfluous intersections in some cells, leading to publishers sending subscribers unneeded data.

There are several ways to deal with the spurious matches that may be detected by the grid-based approach. One method is for the receiver to filter out any unwanted data. Another approach is to allow only the point publications that were described earlier in this section, a technique that reduces the number of superfluous intersections. Finally, an alternative approach to dealing with erroneous detection of intersections is to define a margin of error that is larger than the cell length. Specifically, if the simulation designers define that the accuracy of a dimension is limited to the grid cell length along that dimension, no spurious intersections occur and the need for filtering is eliminated. For example, suppose for a dimension that represents altitude, the grid cell length along that dimension is half a kilometer. If the simulation did not demand greater accuracy than ± 0.5 km when measuring altitude, and if a similar policy existed for all other dimensions, no filtering would be necessary.

In this section we have showed how intersections are determined by the grid-based scheme, that is, by mapping interest regions to cells. The differences among the

various grid-based approaches is due primarily to the way in which they use multicast communication strategies to efficiently transfer data between the hosts that produce data and the hosts that consume it. As noted earlier in this section, the crux of the DDM problem of facilitating data transfer among publishers and subscribers is allocating multicast groups and assigning the hosts to the groups.

The most common grid-based approach to multicast group allocation in DDM is the fixed grid-based [31]. Optimizations to this method include multi-level grid [1] and clustering approaches [3, 7, 18]. We now summarize the strengths and weaknesses of the fixed grid-based strategy, and differentiate it from our alternative approach.

4.3.1 Fixed Grid-Based DDM

Much of the work on DDM has centered on its use within the DoD High Level Architecture (HLA), which defines DDM as one of the six classes of services that are provided to federates by the Run-Time Infrastructure (RTI) [9]. Fixed Grid-Based allocation of multicast groups was the approach used in the initial implementation of the HLA-RTI, which is known as the RTI Prototype [8]. The DDM scheme associated multicast groups with cells that are defined by a grid system overlaid on the terrain [21]. This algorithm has also been used by the light-weight RTI developed at George Mason University [20] and by systems that use the DIS protocol, as opposed to the HLA/RTI [16, 23].

The Fixed Grid-Based scheme is straightforward. First, a multicast group is assigned to each grid cell at system initialization. As the simulation progresses, an RTI component running on each host maps the interest regions of that federate (we assume

one federate per host) to grid cells, and the federate joins the multicast groups that have been pre-assigned to those cells.

Federate	Entity	Interest Type	Cells in Region	Assigned to Group
Fed1	Spy Plane	Subscription	1, 3	MG1, MG3
Fed 2	Squadron A Planes	Publication	3	MG3
Fed3	Squadron B Planes	Publication	4	MG4

Table 3. Fixed Grid-Based Grouping

Table 3 shows how the DDM system maps the grid cells to multicast groups. It also shows which entities join which groups, for the scenario presented in Figure 8. Note that this is a snapshot of the simulation, since membership in any multicast group will change as the publications and subscriptions change. We assume three federates are participating in the simulation, and the plane entities are distributed among them as follow: one of them (Fed1) is simulating the spy plane, and the others (Fed2 and Fed3) are each simulating Squadron A and Squadron B respectively. The spy plane is subscribing to the terrain within its radar range, which is mapped to Cell1 and Cell3. Therefore, Fed1 joins Multicast Group 1 and Multicast Group 3 (MG1 and MG3), that have been assigned to Cell1 and Cell3 respectively. The planes in Squadron A are mapped to Cell3, while those in Squadron B are publishing in Cell4, so Fed2 joins MG3 and Fed3 joins MG4.

A feature unique to the Fixed Grid-Based method is that intersection regions are not detected, even indirectly. The major advantage of this “shortcut” is that inter-federate

communication is drastically minimized. Despite the fact that intersections are not considered by the algorithm, whenever an intersection is present, data will be properly transferred between the publisher and subscriber because they both will have joined the same multicast groups. These groups are the ones that corresponding to the cells in the intersection, which in our example is Multicast Group 3 (MG3). However, data will be transferred for every publication cell, even when no intersection is present. The costs of this irrelevant data transmission must be weighed against the benefits of decreased dependency between hosts to evaluate the appropriateness of the Fixed Grid-Based method for a particular simulation.

4.3.2 Dynamic-Grid Based DDM

As explained in the previous section, the fixed grid-based scheme does not directly determine intersection regions. Therefore, a technique of triggering federates to start (or stop) transmitting (or receiving) on a group, which is used in the region-based method, cannot be used. Thus there is no mechanism to prevent publishers from sending data on a group that no subscribers have joined, such as MG4. Our dynamic grid-based method [4, 5], which we shall describe in the next chapter, addresses this drawback of the fixed-based scheme.

The dynamic-grid based scheme is a hybrid of the fixed-grid based and region-based methods since it reforms the fixed-grid based by giving it the intersection detection and triggering abilities of the region-based scheme. However, the term *hybrid approach* has been used in the literature to refer to a different combination of these two schemes, which we shall describe next.

4.4 Hybrid Approach to DDM

The hybrid approach seeks to reduce the matching cost associated with the region-based approach by only performing matching between publications and subscriptions that are part of intersections [29]. A federation-wide DDM_Coordinator, to which all federates send interest information, determines intersections by mapping the interest regions onto a grid, as in the grid-based method. This is the approach that is being used for the development of the DDM-Kit [14], which in the future will be integrated into the RTI-Kit that we describe in the next section.

The hybrid approach limits the amount of matching that the DDM_Coordinator must perform, which is its fundamental advantage over the region-based scheme. However, the hybrid approach, like the region-based, still requires a federation-wide DDM_Coordinator. The use of a centralized coordinator tends to limit scalability, which is a drawback compared to the fixed grid-based approach, which has no coordinator, and the dynamic grid-based approach, which uses distributed coordinators.

4.5 Comparison of DDM Approaches

We now summarize the strengths and weaknesses of the DDM approaches that we have discussed. The region-based method provides exact matching between publication and subscriptions regions, but the matching is performed by a single, centralized coordinator that can become a bottleneck when several federates are frequently updating their interest regions. The hybrid method reduces the matching cost of the region-based scheme, but does not address the problem of a central coordinator. The fixed grid-based

strategy has no central coordinator, which is its main advantage over the region-based and hybrid schemes, but it does not perform exactly matching and therefore can send unwanted data which must be filter at the source. Another disadvantage of the fixed grid-based method is that it tends to use a much higher number of multicast groups that the other two approaches because it does not determine intersections.

	Region	**Fixed Grid**	**Hybrid**
Exact Matching	X		X
Non-Exhaustive Matching		X	X
No Central Coordinator		X	
Intersection Detection	X		X

Table 4. Comparison of DDM Approaches

Table 4, we provide a comparative study of the DDM schemes described in this paper. The rows denote positive elements of various approaches. *Exact Matching* between publication regions and subscription regions is beneficial because it can eliminate reception of unwanted data. Comparing all publications to all subscriptions is costly, so *Non-Exhaustive Matching* is desirable. A federation-wide DDM_Cooridinator can be a bottleneck, so having *No Central Coordinator* increases scalability. *Intersection Detection* is necessary to prevent publishers from sending data in which no subscriber is interested.

5. DYNAMIC GRID-BASED ALGORITHM

In this chapter, we present a detailed description of our dynamic grid-based algorithm. Like all grid-based approaches, we define the cells by overlaying the terrain within a grid. However, unlike the fixed grid-based scheme which statically assigns multicast groups to all of the cells in the grid, we dynamically allocate multicast groups based on the current publication and subscription regions in the system and then trigger hosts to join those groups, as in the region-based method. Our strategy is unique because only those cells, in which there is at least one federate publishing and at least one federate subscribing, are assigned to a multicast group. In other words, a multicast group is allocated to each cell that has been determined to be part of the intersection of a publication region and a subscription region [4, 5].

Federates join and leave the appropriate groups as the result of trigger messages send by the grid system. Publishers only join and transmit on a group if there is at least one subscriber interested in that data, and subscribers only join and listen on a group if there is at least one publisher transmitting on that group. This technique has the dual advantage of (1) preventing senders from transmitting data needlessly; and (2) reducing the number of multicast groups that a federate needs to join, since groups are only joined and data is only sent when intersections are detected.

For example, consider the scenario from Section 3 that was used to illustrate the fixed grid-based method. Using the dynamic approach, an intersection would be detected in Cell 3, and Fed1 and Fed3 would then be triggered to join MG3, as shown in

Table 5. In our simple example, the dynamic method used only one multicast group (MG3) as compared to the three groups (MG1, MG3, MG4) needed by the fixed method. For a large-scale simulation with many federates and objects, the savings in multicast groups used that the dynamic scheme provides, as opposed to the fixed method, is tremendous, as shown by our experimental results which we describe in later sections. This is an important advantage of the dynamic grid-based approach.

Federate	**Entity**	**Interest Type**	**Cells in Region**	**Assigned to Group**
Fed1	Spy Plane	Subscription	1, 3	MG3
Fed 2	Squadron A Planes	Publication	3	MG3
Fed3	Squadron B Planes	Publication	4	<none>

Table 5. Dynamic Grid-Based Grouping

Another benefit of our scheme is that intersection detection and triggering is performed by RTI components on each host, which may be viewed as distributed DDM_Coordinators. Each one is responsible for keeping track of a specific set of grid cells, as we describe in detail in later sections. There is no central database or coordinator, making our dynamic approach more scalable than a region-based scheme that uses a central DDM_Coordinator.

5.1 Overview of the Dynamic Grid-Based Algorithm

In Figure 9, we present the basic steps our algorithm follows when a federate *F* publishes or subscribes to a region containing cell *Ci*. Recall that if there was previously a publisher and a subscriber whose regions overlapped in the cell *Ci* then there is already a multicast group assigned to that cell. Consequently, federate *F* simply joins that multicast group and performs the final step of our scheme. Otherwise, if the publication or the subscription made by federate *F* creates an intersection between the publisher(s) and subscriber(s) region in that cell, a multicast group is then allocated to that cell. Federate *F*, as well as all of the other entities that are publishing or subscribing in that cell *Ci*, must join that group. In either cases the final step must be carried it out.

The final step in our scheme will record the publication or the subscription of the federate to the cell. This step is necessary because if that cell is not currently assigned to a multicast group, it may be assigned in the future, in which case federate *F* will need to be told to join that multicast group. In Figure 9, note that a "Pub/Sub Match" occurs when the following condition is met: there is at least one publisher, and at least one subscriber, in the cell.

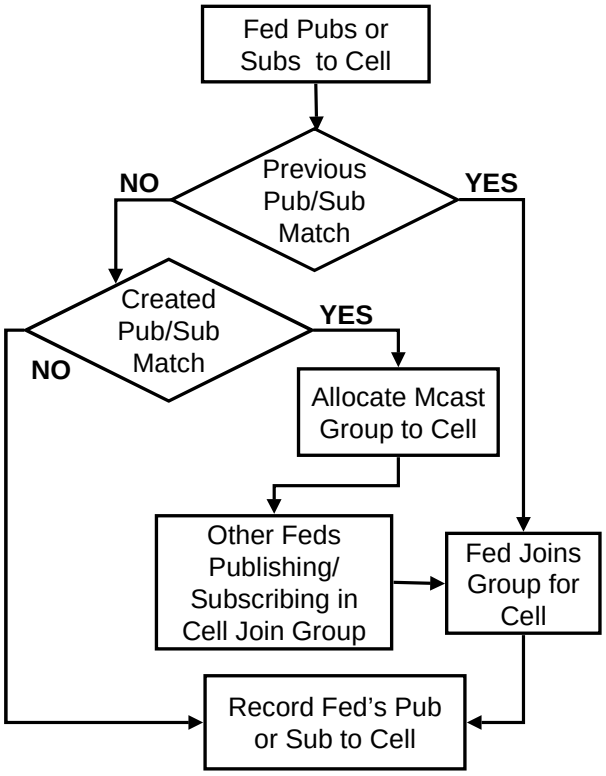

Figure 9. Overview of the Dynamic Grid-Based Algorithm

5.2 Implementation Details

The GRID system implements our Dynamic Grid-Based approach to Data Distribution Management by performing the following tasks.

TA) Creating a distributed grid.

Cells are evenly distributed among the nodes that are participating in the simulation. The node (or CPU) to which a cell is distributed is said to be the owner of that cell. We make use of the card dealer strategy to map cells to nodes; i.e.,

$$Owner = Ncell \ \% \ Nodes;$$

where *Owner* represents the cell *owner*, *Ncell* indicates the cell number, and Nodes is the total number of the nodes participating in the simulation.

Our system supports the use of multiple independent grids, and thereby can be used to represent different routing spaces. Thus, both cell number and Grid_ID must be provided to specify a cell. The number of grids being used, the number of dimensions in each grid, and the size of each dimensions is stored on each node. Subscription and publication information for a particular cell Ci, i.e., the list of nodes that are publishing or subscribing to Ci, is stored in the node that owns that cell.

TB) Keeping track of federates that are publishing and subscribing to each cell.

For each cell owned by a node (owner), we make use of two bit_arrays to identify the nodes that are respectively publishing and subscribing to that cell. Thus, if node i is publishing or subscribing to that cell, the ith bit in the array is set. Similarly, unpublishing or unsubscribing clears the bit. When a node publishes or subscribes to a cell owned by another node, the GRID mechanism sends a message to the owner node, requesting that the publishing (or subscribing) node be recorded using the corresponding bit_array. Messages are sent and received using a call to the FM library. Unsubscribing and unpublishing are done in a similar way.

TC) Allocating groups to cells where publisher/ subscriber intersection occurs.

When a cell owner adds a publisher or a subscriber to a cell where there is already both a publisher and a subscriber present, it does a lookup to find the multicast group assigned to that cell. If the owner adds a publisher to a cell where there was previously only subscribers, or visa versa, it then makes a call to an MCAST library routine. The

routine allocates a new multicast group and returns it to the cell owner, who assigns it to the cell and becomes the group manager, administering its membership list.

TD) Notifying federates when they need to join or leave the groups.

Once the cell owner detects a publisher/subscriber intersection in a cell and retrieves or creates the multicast group assigned to that cell, all nodes publishing or subscribing to that cell must be told to join the group. The owner sends them a join_group message using calls to the FM communications library. Similarly, when the owner detects the cessation of a publisher/subscriber intersection due to unpublishing/unsubscribing, it sends a leave_group message.

TE) Triggering federates to join or leave the groups.

When a node receives the join_group or leave_group message sent in task TD, it then makes a call to an MCAST library routine. The routine subscribes or unsubscribes that node to the group, and then it sends an update message to the node that is the manager of the multicast group. This message instructs the group manager to update the group membership list by invoking the MCAST library.

In Figure 10, we present the function, performed by the cell owner, that adds a publisher to the cell and completes tasks TB, TC, and TD.

```
// Add a new publisher, which lives on the given node,
// to the given cell

OwnerAddPub (Grid_ID, Cell_ID, Node_ID)
{
 LocalID = GetLocalIndex (Cell_ID);
 prev_match = PublishersHere(Local_ID) &&
SubscribersHere(Local_ID);
// add this node as publisher in cell
  AddPub(GridID, Local_ID, Node_ID);
 curr_match =PublishersHere (Local_ID) &&
SubscribersHere(Local_ID);
 // Check if there are any intersections
  if  (! curr_match)  return() ;
  if  (prev_match)
  {  // group already assigned, lookup and tell Pub to join
          GroupsOwnedLookup (Grid_ID,  Cell_ID, &Handle);
         OwnerTellJoin(Grid_ID, Cell_ID, Handle, Node_ID);
   }  else {
          // new intersection has been created
          Name = MakeName(Grid_ID,Cell_ID);
          CreateMcastGroup(Name,&Handle);
         // store group's handle in table
         GroupsOwnedInsert (Grid_ID, Cell_ID, Handle, Name);
         // tell other publishers or subscribers to join
         for  (i=0; i<NUM_NODES; i++)
           {
              if (PublisherHere(Local_ID, i) ||
                  SubscriberHere(Local_ID,i))
                 OwnerTellJoin(Grid_ID, Cell_ID, Handle, i);
            }
      }
}
```

Figure 10. Cell Owner Pseudocode

5.3 Grid-Kit

The algorithm in Figure 9 and the corresponding pseudocode in Figure 10 are carried out by our Grid-Kit, which RTI builders can use to accomplish Data Distribution Management. In order to test the Grid-Kit, we have developed a UNT-RTI that uses the Grid-Kit to perform DDM. The UNT-RTI does not perform all of the services specified in by the High Level Architecture, and is not meant to be a complete RTI. In this section we will describe how the UNT-RTI uses the Grid-Kit to carry out DDM.

Recall that a Grid-Kit component is present on each node on which the UNT-RTI is running. Since the UNT-RTI is linked to each federate executable program, the UNT-RTI, and hence the Grid-Kit, runs on all nodes that are participating in the simulation. (We assume one federate per node.) One of the primary functions of the Grid-Kit is to represent the grid cells, and these cells are distributed among the Grid-Kit components running on each node. There is no centralized representation of the grid, which is one of the strengths of our approach.

Note that both the cells being managed by the Grid-Kit and the objects being simulated by the federate reside on each node. However, the federate does not "know about" the grid cells, since a federate should not make assumptions about how an RTI is performing DDM. For example, a federate should not care whether an RTI is using the Grid-Kit to perform dynamic grid-based DDM or using another mechanism to perform region-based DDM. Just as the federate should be designed to work with different RTIs, we have designed the Grid-Kit to be easily integrated into different RTIs. For this

reason, the Grid-Kit does not "know about" individual objects, only federates. The RTI is responsible for managing information about the objects being simulated by each federate, if desired, and this task is indeed performed by our UNT-RTI.

Federates express interest in terms of regions. Next, the UNT-RTI translates these regions into the cells they overlap. For each cell, the UNT-RTI sends a request to the Grid-Kit that is managing that cell, directing it to publish, subscribe, unpublish, or unsubscribe that federate to or from that cell. The interface between the Federate, UNT-RTI, and Grid-Kit on a particular node is shown in Table 6. The Grid-Kit is technically part of the UNT-RTI, although in this context the term *UNT-RTI* refers to all UNT-RTI components except those that are part of the Grid-Kit. The term *Operation* represents one of the following: Publish, Subscribe, Unpublish, Unsubscribe.

	Federate	**UNT-RTI**	**Grid-Kit**
Responsibility	simulates objects	maps regions to cells	manages cells
Steps	1) object performs *operation* with region	2) performs *operation* for each cell in region	3) updates data structures based on *operation*

Table 6. Component Interface

As illustrated in Table 6, the federate and UNT-RTI involved in steps 1 and 2 are both on the same node, assume it is Node X. However, the Grid-Kit that performs step 3 is not necessarily on Node X. In other words, the region used in step 1 will be mapped to one or more cells in step 2, and these cells are either managed *locally*, by the Grid-Kit on

Node X, or *remotely*, by the Grid-Kit on other nodes. For remote cells, the UNT-RTI on Node X will send a message to the remote node requesting the appropriate operation be performed on the specified cell. For local cells, the UNT-RTI simply makes a function call that invokes the local Grid-Kit.

5.4 Federation Example

We shall now take the simple Spy Plane federation as an example to illustrate how the Grid-Kit operates. Recall that there are three federates participating in this scenario, and each one is simulating one or more objects.

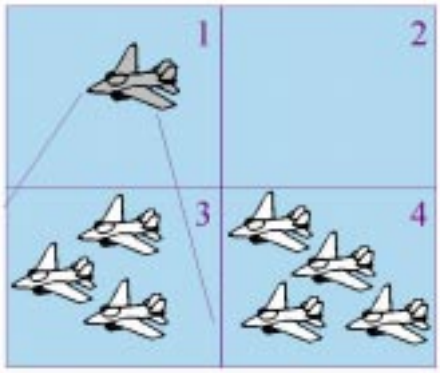

Figure 11. Grid Overlaid

5.4.1 Create grid cells

During the simulation initialization phase, the grid cells are created. In this example, we consider only the four cells, as shown in Figure 11. We assume they are distributed to the Grid-Kit on each node (we assume exactly one federate per node). Cells are initially set to be "empty," and as the simulation progresses they will be updated to reflect the interests of the federates. The total number of grid cells remains constant

throughout the simulation, since cells are only created during initialization, and they are only deleted when the simulation is complete.

Table 7 shows the objects and grid cells that are being simulated by the federates on each node. The thick lines between the Grid-Kit column and the federates columns are intending to emphasize the independence of the two. In other words, there is not necessarily any relationship, in term of the scenario being simulated, between the Spy Plane and cells 1 and 4, even though that object and these cells are all being represented on node 1. Furthermore, Fed1 has no knowledge of the cells, and the Grid-Kit has no knowledge of the Spy Plane.

Node	**Grid-Kit**	**Federates**	
	Manages Cell(s)	**Name**	**Simulates Object(s)**
1	1, 4	Fed1	Spy Plane
2	2	Fed2	Squadron A Planes
3	3	Fed3	Squadron B Planes

Table 7. Distributed Grid

5.4.2 Federates Express Interest

Continuing with our Spy Plane scenario, suppose the initial positions of the planes are as shown in Figure 8. The sequence in Table 5.4 shows how Fed1 will subscribe to the region representing the Spy Plane's radar range. Note that Fed1 can subscribe to any grid cells, in this case it selects cells 1 and 3, not only the ones being representing by the

Grid-Kit on the same node as Fed1, which in this case are cells 1 and 4, as shown in Table 7.

	Fed1	**UNT-RTI**	**Grid-Kit**	
Node	1	1	1	3
Steps	1) Spy Plane *Subscribes* to region	2a) Requests *Subscribe* Fed1 to cell 1 2b) Requests *Subscribe* Fed1 to cell 3	3a) updates cell 1 with Fed1 *Subscription*	3b) updates cell 3 with Fed1 *Subscription*
Messages	<none>	Sends to node 3	<none>	Receives from node 1

Table 8. Spy Plane Subscription

Our Spy Plane scenario highlights the difference between subscribing to a local cell and a remote cell. Step 2a in Table 8 illustrates how the UNT-RTI on node 1 requests a subscription to a local cell, managed by the Grid-Kit on node 1. On the other hand, in step 2b the UNT-RTI requests a subscription to a remote cell, being managed by the Grid-Kit on node 3, by sending a message to it. In general, an operation on a local cell will be performed more quickly than on operation on a remote cell, because of the latency involved in sending a message between nodes.

Next, the planes in Squadron A will publish in cell 3, as shown in Table 9.

	Fed2	UNT-RTI	Grid-Kit
Node	2	2	3
Steps	1) Squadron A Planes *Publish* to region	2) Requests *Publication* of Fed2 to cell 3	3b) updates cell 3 with Fed2 *Publication*
Messages	<none>	Sends to node 3	Receives from node 2

Table 9. Squadron A Planes Subscription

The Steps in Table 9 shows how the UNT-RTI makes publications and subscriptions on behalf of the federate, not individual objects. Each of the planes in Squadron A will publish, but each of the their publication regions will be mapped to cell 3, so the RTI only requests *one* publication to that cell. The UNT-RTI does keep track of which local objects are subscribing to which cells, and although the Grid-Kit is not given this information, the UNT-RTI needs to have it in order to determine when to unpublish from cell 3. For example, if one of the Squadron A planes, let us call it A1, were to suffer battle damage and change course, meaning Fed2 would request that the A1 object be unpublished (and perhaps published with a different region), the UNT-RTI would not want to unpublish Fed2 from cell 3. The other Squadron A planes (A2, A3, and A4) are still publishing in that cell 3, so the Grid-Kit managing cell 3 would not be contacted. The UNT-RTI would simply record that A1 is no longer publishing in cell 3. The UNT-RTI would only unpublish Fed2 from cell 3 after objects A2, A3, and A4 had also been unpublished from the region that overlaps cell 3.

The publication of the Squadron B planes is similar to Squadron A, as shown in Table 10.

	Fed3	**UNT-RTI**	**Grid-Kit**
Node	3	3	1
Steps	1) Squadron B Planes *Publish* to region	2) Requests *Publish* Fed3 to cell 4	3b) updates cell 4 with Fed3 *Publish*
Messages	<none>	Sends to node 1	Receives from node 3

Table 10. Squadron B Planes Subscription

5.4.3 Detect Intersection and Assign Multicast Group

The Grid-Kit records the interests of each federate, and Table 11 shows the publications and subscriptions that have been made up to this point in the scenario. The Grid-Kit on node 3 detects an intersection in cell 3 between the publication of Fed2 and the subscription of Fed1, and assigns a multicast group to that cell. The name of the group is "$MG_{x\text{-}i}$" signifying that it is the i^{th} group created by the Grid-Kit on Node X, which is this case is $MG_{3\text{-}1}$.

	Grid-Kit			
Node	1		2	3
Cells	1	4	2	3
Publishers	<none>	Fed3	<none>	Fed2
Subscribers	Fed1	<none>	<none>	Fed1
Assigns Group	<none>	<none>	<none>	$MG_{3\text{-}1}$

Table 11. Publications and Subscriptions

The row in Table 11 labeled “Node” represents the host of the Grid-Kit, although the federates that are publishing or subscribing to cells managed by that Grid-Kit can be on different nodes. The Grid-Kit knows the mapping between federates and the nodes on which they are hosted, which is necessary for determining to which nodes the messages triggering federates to join groups should be sent.

5.4.4 Trigger Federates to Join Group

The final Steps of this example is shown in Table 12. After and intersection as been detected, trigger messages are sent to the interested federates. More specifically, the messages are send from the Grid-Kit that detected the intersection to the UNT-RTI running on the nodes of the federates that are being triggered. The UNT-RTI then joins the given multicast group. In earlier sections we have stated that the federates join multicast groups, but in practice it is the UNT-RTI that joins and leaves groups since the

federates are not aware of the underlying communication mechanism that is being used, in this case multicast.

	Grid-Kit	**UNT-RTI**	
Node	3	1	2
Steps	1a) Triggers Fed1 to Join MG_{3-1} for cell 3 1b) Triggers Fed2 to Join MG_{3-1} for cell 3	2a) Joins MG_{3-1}	2b) Joins MG_{3-1}
Messages	Sends to node 1 and 2	Receives from node 3	Receives from node 3

Table 12. Trigger Federates to Join Group

Once the UNT-RTI on nodes 1 and 2 have joined MG_{3-1}, a communication channel is formed between Fed1 and Fed2. Thus, Fed1 will receive updates from Fed2, and the Spy Plane will be able to monitor the location of the Squadron A planes.

6. SIMULATION EXPERIMENTS

In this chapter, we will report the simulation experiments, which we have conducted to evaluate the performance of our dynamic grid-based approach. We will also present the comparison of our approach to the fixed grid-based method.

6.1 Hardware Platform and Federation Scenario

The platform for our experiments is a Beowulf cluster of ten 300 MHz Pentium II computers connected by 100Mb Ethernet, running Linux. In our simulation runs, there is one federate per node. Each federate simulates the same number of objects, in this case, tanks.

Parameter	**Value**
Number of Federates	10
Terrain Size (km^2)	500
Number of Routing Space Dimensions	3
Number of Time-Steps	100

Table 13. Constant Parameters

Table 13 summarizes the parameters that were held constant throughout all our experiments. The scenario used is known as Tank Dogfight, and is based on the scenario used in [16]. The objects are randomly placed somewhere on the two-dimensional

terrain. The terrain area is 500 km^2. At each time-step, for each tank a direction (North, South, East, or West) is chosen at random and that tank moves a pre-determined distance in that direction, which we call the Move_Distance. Tanks use point publications, meaning they publish to exactly one cell at a given time-step. Tanks do not "fall off" or "wrap-around" the edge of the terrain, but rather hold their position. For example, if a tank has moved as far westward as the terrain allows, and another movement West is randomly chosen, the tank will not move during that time-step. Tanks subscribe to a square region, centered at their current position. The square sides have length Subscribe_Distance. Simulations consisted of 100 time steps.

Dimension	**Type**	**Name**	**Represents**
1	Spatial	N-S	North-South position in terrain
2	Spatial	E-W	East-West position in terrain
3	Non-spatial	Team	Red or Blue Alliance

Table 14. Routing Space Dimensions

Now let us specify the characteristics of the routing space used in our experiments. Table 14 illustrates the routing space dimensions for the Tank Dogfight. Five federates are designated as belonging to the Red Team and the other five are member of the Blue Team. Red Team tanks only subscribe to Blue Team tanks, and Blue Team tanks are only interested in tanks from the Red Team. Therefore, the routing space

has three dimensions: North-South, East-West, and Team. Tanks only move in the two spatial dimensions, since they do not change teams during the simulation.

6.2 Performance Metrics

It is important to recall that the additional overhead incurred by our scheme, as opposed to the Fixed Grid-Based one, is due to the need to create multicast groups as the simulation progresses and trigger federates to join those groups. Thus, to assess the overhead of our scheme, we choose the following three metrics.

DDM Time: indicates the total time consumed by the DDM service. It includes the time needed to publish or subscribe to a cell or a region (i.e., group of cells), and the time needed to create any multicast groups, trigger entities to join, and update all necessary tables. It also includes the time consumed by the triggered entities (or federates) in joining the appropriate group.

DDM Messages : represents the total number of messages generated to perform the task of publishing/ subscribing to a cell or a region. Thus

$$DDM\_Messages = TotMsg\_GRID + TotMsg\_MCAST;$$

where *TotMsg_GRID* and *TotMsg_MCAST* represent the total number of messages generated by the GRID and the MCAST libraries respectively. The first type of message is used to notify a cell that a node needs to be added to its publisher (or subscriber) list. The second type of message is used to coordinate the creation of the multicast group, and to update its memberships.

Multicast Groups Used: the number of groups created during the simulation.

6.3 Experimental Methodology

We have conducted four groups of experiments. The purpose of each experiment can be described as follows. First, we will present the results obtained for Set A and Set B experiments. Next, we will discuss the Set C and D experiments which we carried out to compare our approach to the fixed grid-based DDM method.

Set A Experiments – assess the performance of our dynamic grid-based strategy for a large number of grid cells

Set B Experiments – assess the performance of our dynamic grid-based strategy for a large number of objects

Set C Experiments – compare our approach to the fixed grid-based method

Set D Experiments – further investigate results observed in Set B

6.3.1 Sets A and B: Evaluating Our Approach

Our aim in Sets A and B experiments is to determine the scalability of our approach by studying our dynamic grid-based scheme for increasing number of objects and cells. The parameters that were held constant for all runs of these experiments included the ones listed in Table 13, as well as Move_Distance and Subscribe_Distance, which were set to 4 km and 8 km, respectively.

Experiment Set A tests our system for increasing number of grid cells. The total number of objects is held constant at 1000. The parameter that determines the number of grid cells is the number of units per dimension. In our scenario the Team dimension always has two units, since there are only two values along this dimension, Red and Blue.

We also assume that the first spatial dimension, N-S, always has the same number of units as the second spatial dimension, E-W. Changing the number of Units Per Spatial Dimension (UPSD) affects the mapping of the terrain onto the grid, although the terrain area being represented remains the same.

The total number of cells in the grid can be calculated by multiplying the number of units for all three dimensions. The number of units in each of the two spatial dimensions is represented by the UPSD, and the number of units in the non-spatial Team dimension is always two. Thus the, the total number of grid cells equals to UPSD $\times$ UPSD $\times$ 2. In our experiments, we varied the UPSD from 200 to 1,000 (i.e. from 400,000 to 2 million grid cells). In Table 15, we show the size of the terrain area to which a single cells is mapped for different values of UPSD. Recall that the total terrain area for our simulation is 500 km^2.

UPSD	**200**	**400**	**600**	**800**	**1000**
Terrain Area (km^2) Represented by 1 cell	2.50	1.25	0.83	0.63	0.50

Table 15. Cell to Terrain Area Mapping

In Set B experiments, we investigate the performance of the GRID system while varying number of total objects in the simulation. Note that objects are still distributed evenly among the nodes. UPSD is held constant at 500. The UPSD was kept constant at 1000, yielding 2,000,000 grid cells (i.e. UPSD $\times$ UPSD $\times$ 2).

6.3.2 Sets C and D Experiments: A Comparison to the Fixed-Grid Approach

In Set C and D experiments, we compare our dynamic grid-based method to the fixed grid-based scheme. For these tests, we held the number of objects constant at 1000, and we vary the UPSD up to 100.

	Region	Fixed Grid	**Dynamic Grid**
Exact Matching	X		
Non-Exhaustive Matching		X	**X**
No Central Coordinator		X	**X**
Intersection Detection	X		**X**

Table 16. Comparison of Dynamic to Other DDM Approaches

Table 16 summarizes the strengths (indicated by an X in the corresponding row) of the region-based, fixed grid-based, and dynamic grid-based strategies. Recall that the main advantage of our dynamic grid-based approach, when compared to the fixed grid-based, is that our method assigns multicast groups only to those cells that are part of an intersection between a publication and subscription regions. This strategy is accomplished by using the triggering technique that is employed by the region-based method of DDM. The fixed grid-based scheme allocates groups to cells that are part of either a publication or a subscription region, meaning that a significantly higher number of multicast groups are used as compared to our dynamic grid-based approach.

As shown in Table 16, the fixed grid-based scheme does not detect intersections. However, for our dynamic method, each time a publisher/subscriber match is detected in a cell, our GRID system triggers all federates that are either publishing or subscribing in that cell to join the multicast group associated with that cell. If the intersection in that cell is later nullified because of the unpublish or unsubscribe request of one or more federates, all federates that are members of the multicast group will be triggered to leave that groups.

The relationship between the number of intersections detected and the number of intersections nullified, during the course of a particular experiment, is an important parameter that we investigate in Set D experiments. We can calculate the relationship between intersection detection and intersection nullification according to the equation

$$Percent\_Nullified = \frac{TID}{TIN} \times 100,$$

where *TID* is Total Number of Intersection Detected, and *TIN* represents Total Number of Intersections Nullified.

Generally, intersections are detected at a certain time-step, but then undone in a later time-step as the objects move around the terrain, so the percentage of intersections nullified is usually quite high, meaning over 95%.

6.4 Experimental Results

In order to study the performance of our algorithm, we have designed four sets of experiments, as described in the previous section. In the first two sets, we assess the performance of our approach, and in the second two sets of experiments we compare the results obtain with our method to the results obtained using the fixed grid-based strategy.

6.4.1 Set A Experiments

Figure 12 show the resulting values for the DDM_Time, and Figure 13 portrays the values obtained for DDM_Messages, as a function of the number of UPSD. The increase in both parameters is similar to a linear pattern. These results clearly indicate that for up to 2 million grid cells, our scheme scales in terms of DDM_Time and DDM_Messages.

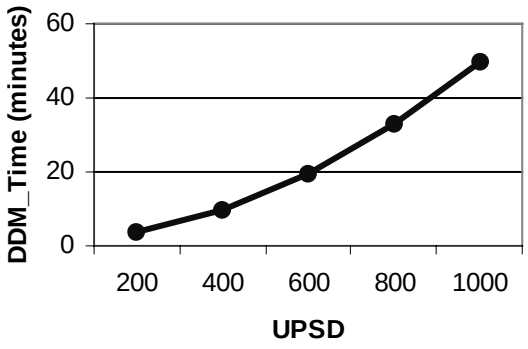

Figure 12. DDM_Time vs. UPSD

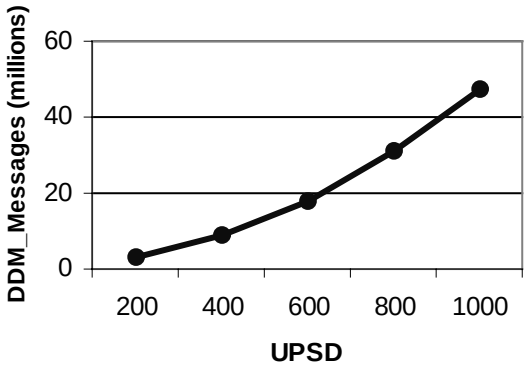

Figure 13. DDM_Messages vs. UPSD

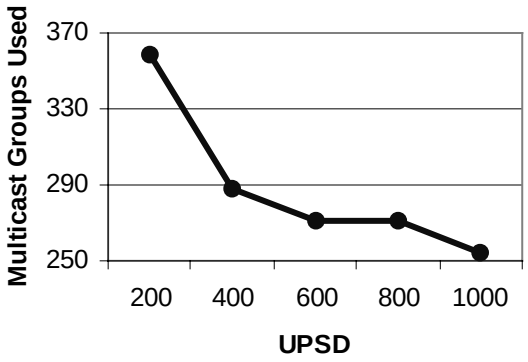

Figure 14. Number of Multicast Groups vs. UPDS

Figure 14 portrays the Number of Multicast Groups Used as a function of the number of UPSD. Note that the Multicast Groups Used decreases slightly as the UPSD is increased. This behavior is not surprising, since as the number of cells increases, the cell size (in terms of the terrain area it represents) decreases, allowing greater accuracy in detecting intersections. Fewer intersections are detected, so fewer multicast groups are used. In the curve of Figure 14, we note a sharp decrease in multicast group usage, meaning increased accuracy, between 200 and 400 UPSD. For greater than 400 UPSD,

the decrease is minimal, suggesting that for this scenario, using at least 400 UPSD is desirable for accurate intersection detection.

6.4.2 Set B Experiments

Set A experiments assessed the performance of our approach using scenarios with a large number of *grid cells*. Experiment Set B evaluates our scheme for scenarios with a large number of *objects*.

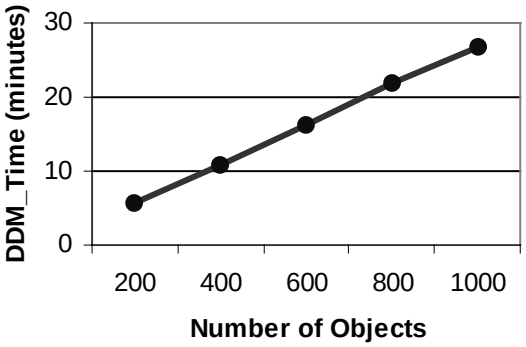

Figure 15. DDM_Time vs. Number of Objects

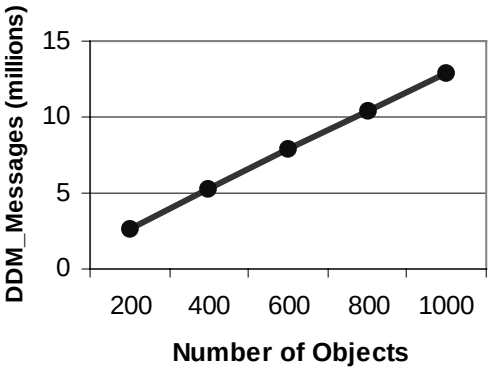

Figure 16. DDM_Messages vs. Number of Objects

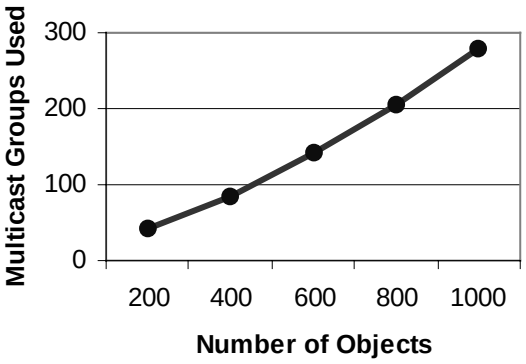

Figure 17. Number of Multicast Groups Used vs. Number of Objects

Figures 15 – 17 portray the values obtained for the DDM_Time, DDM_Messages, and Multicast Groups Used as a function of the total number objects employed in the simulation model. As we can see from the curves in Figure 15 and Figure 16, which are similar to those obtained in our first set of experiments, the DDM_Time and DDM_Messages increase according to a linear pattern as we increase the number of objects, showing the scalability of our system. igure 17 illustrates that the Multicast Groups Used also increases as the number of objects grows. Such an increase is to be expected, since as more objects are involved in the simulation, more intersections will be detected.

6.4.3 Set C Experiments

In the previous section, we reported the results from Experiment Sets A and B, which clearly showed that our scheme is scalable for scenarios with a very large number of cells and objects. In this section we compare the performance of our dynamic grid-based approach to the fixed grid-based method and describe the results from Experiment Set C.

In these experiments, the maximum UPSD is 100. The reason why we do not present comparison results for higher UPSD is that for UPSD greater than 100, the fixed grid-based performs very poorly in comparison to the dynamic approach or fails to complete the simulation altogether [19]. At 1000 UPSD the fixed method can only simulate up to 30 objects before exhausting the available memory. The high memory requirements of the fixed grid-based scheme are due to the huge number of multicast groups that must be created, as shown in Table 17.

	Fixed Grid-Based			Dynamic Grid-Based		
Objects	*DDM_Time*	*DDM_*	*Groups*	*DDM_Time*	*DDM_*	*Groups*
10	9.45	962,435	56,056	1.09	494, 454	0
20	13.14	1,848,609	98,860	2.11	994,140	0
30	24.68	2,994,995	149,083	3.18	1,491,699	1
40	<system memory exhausted>			4.33	1,944,024	4

Table 17. Fixed vs. Dynamic Grid-Based

For our next set of experiments, we decreased the UPSD to a level at which the fixed grid-based approach would not exhaust the system memory. As with the previous set of experiments, the Move_Distance and Subscribe_Distance were held constant at 4 and 8, respectively. We varied the number of grid cells, and discovered that for greater than 40 UPSD, the dynamic grid-based approach clearly outperformed the fixed grid-based in terms of all three of our experimental parameters, DDM_Time, DDM_Messages, and Number of Multicast Groups Used.

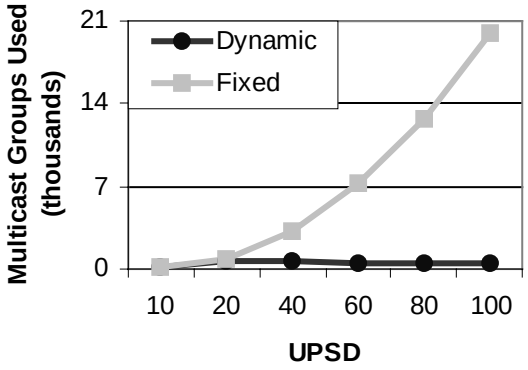

Figure 18. Number of Multicast Groups Used vs. UPSD

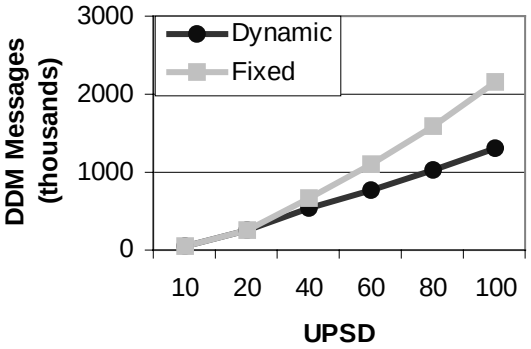

Figure 19. DDM_Messages vs. UPSD

For less than 40 UPSD, our scheme used significantly fewer multicast groups than the fixed grid-based method, as seen in Figure 18. These results clearly indicate that our approach achieves our goal of significantly reducing the number of multicast groups used, as compared to the fixed grid-based approach. In addition, our scheme also sent fewer DDM_Messages, as shown in Figure 19. These results show that the impact of the additional number of trigger messages sent by the dynamic method is less significant than the additional number of messages needed by the fixed method to initialize and join a higher number of multicast groups.

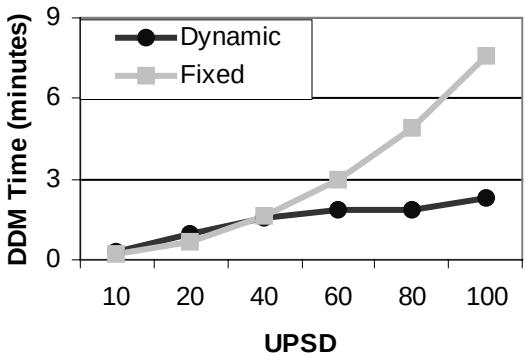

Figure 20. Number of Multicast Groups Used vs. UPSD

On the other hand, Figure 20 shows that for less than 40 UPSD the fixed grid-based method has lower DDM_Time then the dynamic scheme. In attempting to determine the reason why the fixed grid-based scheme had faster DDM_Time when the UPSD dropped below 40, we made the following observation. For less than 40 UPSD the number of times the dynamic grid-based scheme detected an intersection heavily outweighed the number of times it subsequently identified that the intersection had been nullified, meaning the publisher and subscriber had moved away from each other.

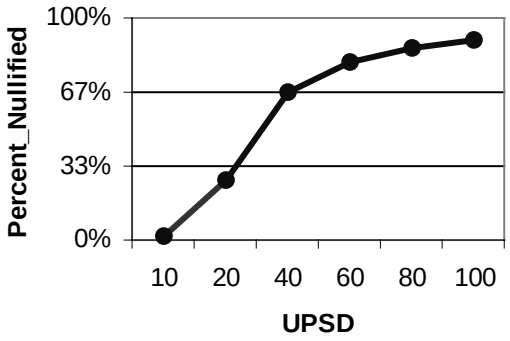

Figure 21. Percent_Nullified vs. UPSD

We see in Figure 21 that the Percent_Nullified for UPSD greater than 40 is over 80%, which is what we would expect in a scenario where all objects are continually moving, as in a Tank Dogfight scenario. However, for UPSD below 40, the percentage of intersections nullified sharply dropped. This is mainly due to the fact that at low UPSD, cells are mapped to a large terrain area. Since the objects cross the cell boundaries less often, intersections that are detected tend to stay in effect for more time-steps than they would if the objects were moving frequently between cells, as they do at high UPSD.

6.4.4 Set D Experiments

In our next set of experiments, we wish to verify our hypothesis about the reason why the dynamic grid-based approach has a slower DDM_Time when compared to the fixed grid-based when we use less than 40 UPSD. We believe that this phenomenon is caused by the tendency of objects to infrequently move between cells when the UPSD drops below a certain threshold, in this case 40 UPSD. We attempted to increase this

threshold by adjusting the scenario parameters so that the simulated objects will move slowly, i.e., they will tend to stay in one cell for a greater number of time-steps. Thus, we decreased the Move_Distance of the objects from 4 km to 1 km. As we expected, the threshold at which the fixed grid-based scheme has a faster DDM_Time than our approach increased from 40 to 60 UPSD, see Figure 22.

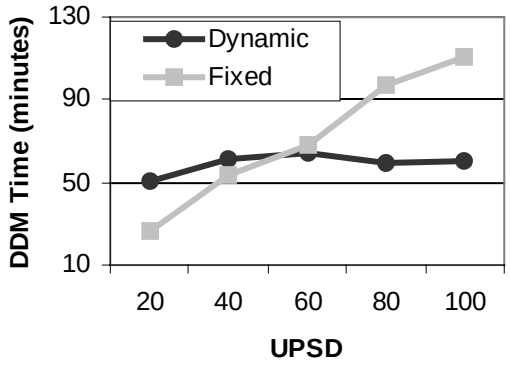

Figure 22. DDM_Time vs. Units Per Spatial Dimension

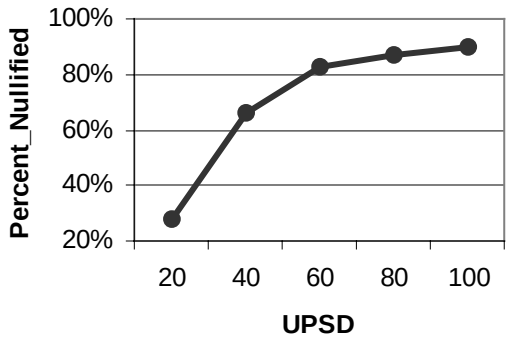

Figure 23. Percent_Nullified vs. Units Per Spatial Dimension

The curve portraying the percentage of intersections nullified in the dynamic grid-based approach, as depicted in Figure 23, steeply declines when the UPSD drops below 60. This behavior is what our hypothesis predicts, since for UPSD less than the threshold of 60 the dynamic approach is outperformed by the fixed, in terms of DDM_Time. Once the curve flattens, in this case for greater than 60 UPSD, the dynamic method has faster DDM_Time than the fixed scheme, as we illustrated in Figure 22. Thus, we verified that there is a relationship between the frequency at which objects move between cells, represented by the percentage of intersection nullified, and the performance of our dynamic approach versus the fixed scheme.

6.5 Summary

In this chapter we have presented the results obtained in four sets of simulation experiments. Sets A and B experiments clearly showed that our dynamic grid-based approach is scalable when we use a large number of grid cells or a large number of objects. The results of Sets C and D experiments indicate that our DDM algorithm used fewer multicast groups and significantly reduced the number of DDM_Messages sent, when compared to the fixed scheme. A significant reduction in DDM_Time was also observed, with the exception of when UPSD was very low.

Based on these results, we conclude this chapter with the following note. In order to take full advantage of the benefits provided by the dynamic grid-based approach, either the UPSD or the Move_Distance should be large enough to ensure that moving objects do not remain in the same cell for an extended number of time-steps. If the simulated

objects are stationary or they are moving very slowly, the fixed grid-based may provide a faster DDM_Time than our dynamic method. However, even in those types of scenarios, our dynamic approach yields a lower or equivalent number of DDM_Messages and significantly fewer Multicast Groups Used, when compared to the fixed grid-based method.

7. CONCLUSION

In large-scale distributed simulations, it is important to reduce the number of messages that are exchanged between the simulated entities. The aim of Data Distribution Management (DDM) is to deal with this issue. A few DDM schemes have been proposed by now. However, commonly accepted approaches have not appeared as yet. In this thesis we focused upon the DDM problem in large-scale distributed simulations.

7.1 Accomplishments

We now summarize the accomplishments of this thesis.

- We have proposed a new approach to DDM that we refer to as a *hybrid* strategy because it used both the trigger messages of region-based schemes and a distributed grid of grid-based approaches.

- We have developed and implemented a lightweight UNT-RTI, which takes advantage of a MCAST-library that performs multicast group management, that we used to study the performance of our DDM scheme.

- We have implemented a fixed grid module, which carries out the fixed grid-based strategy for DDM comparison purposes.

- Empirical performance evaluation of our DDM scheme using large-scale and real-world scenarios showed that a significant reduction in multicast groups used and DDM message overhead over the fixed grid-based approach can be obtained. A 40% reduction in DDM message overhead and a 98% reduction in multicast group usage were observed when compared to the fixed grid-based scheme.

7.2 Directions for Future Research

The research described in this thesis has a number of possible extensions. We describe several problems that might be interesting to work on.

- Extensive simulation experiments and comparative studies of our scheme with the region-based and agent based DDM schemes would be worthwhile to investigate using our UNT-RTI framework.

- Time Management is another important and challenging problem in large-scale distributed simulations. Time Management will help to control, coordinate, and synchronize the progress of the simulation hosts along the federate time-axis [11].

- Currently, there are no visual or graphical tools that display cells represented by an RTI and the movement of federates during the simulation. Such tools might allow the user to visualize the routing space of the scenario being performed. For example, in our Tank Dogfight Scenario, by visualizing the two

routing space dimensions, the user could see which tanks where located in which cells, and thereby get a good picture of the battlefield being simulated. The distributed nature of the grid abstraction is the main challenge in implementing this functionality.

REFERENCES

[1] H. Abrams, K.Watsen, and M.Zyda. "*Three-Tiered Interest Management for Large-Scale Virtual Environments*." ACM Symposium on Virtual Reality Software and Technology, 1998.

[2] J. Banks. "*Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*." John Wiley & Sons, New York, 1998, pp. 3-15.

[3] A. Berrached. "*Alternative Approaches to Multicast Group Allocation in HLA Data Distribution Management*." 98S-SIW-184. Spring Simulation Interoperability Workshop (SIW), March 1998.

[4] A. Boukerche and A. J. Roy, "*In Search of Data Distribution Management in Large-Scale Distributed Simulations."* Summer Simulation Conference, 2000, pp. 12-19.

[5] A. Boukerche and A. J. Roy, "*Dynamic Grid-Based Multicast Group Assignment in Data Distribution Management."* Distributed Simulation and Real-Time Applications Workshop, 2000, pp. 27-34.

[6] J. O. Calvin, C. J. Chiang, S. M. McGarry, S. J. Rak, D. J. Van Hook. "*Design, Implementation, and Performance of the STOW RTI Prototype (RTI-s*)." 99S-SIW-019, Spring Simulation Interoperability Workshop, 1999.

[7] J. O. Calvin and D. J. Van Hook, *"AGENTS: An Architectural Construct to Support Distributed Simulation*," 94-11-142, 11rd Workshop on Standards for the Interoperability of Distributed Simulations (DIS), Sept. 1994.

[8] J. O. Calvin and R. Weatherly. "*An Introduction to the High Level Architecture (HLA) Runtime Infrastructure (RTI).*" 96-14-103, 14th Interoperability of Distributed Simulations, 1996.

[9] J. S. Dahmann, K. L. Morse. "High Level Architecture for Simulation: An Update." 3rd IEEE Distributed Interactive Simulation and Real-Time Applications, 1998, pp. 32-40.

[10] Defense Modeling and Simulation Office (DMSO). "*RTI Development History*." 2000, http://hla.dmso.mil.

[11] Defense Modeling and Simulation Office (DMSO). "*Level Architecture Interface Specification.*" Version 1.3, 1998, http://hla.dmso.mil.

[12] K. E. Frosch and J. M. Pullen. "*Design And Prototype of A Dual-Mode Multicast Application Gateway*". 97S-SIW-121, Spring Simulation Interoperability Workshop 1997.

[13] R. T. Fujimoto. "*Parallel and Distributed Simulation Systems*." John Wiley & Sons, New York, 2000, pp. 3-14.

[14] R. T. Fujimoto, T. Mclean, K. Perumalla and I. Tacic. "*Design of High Performance RTI Software*." IEEE Distributed Simulation and Real-Time Applications 2000, pp. 41-49.

[15] George Mason University Center of Excellence in Command, Control, Communications, and Intelligence. "*Selectively Reliable Transport Protocol*." 2000, http://bacon.gmu.edu/c3i/index.html.

[16] M. Macedonia, M. Zyda, D. Pratt, and P. Barham. "*Exploiting Reality with Multicast Groups: a Network Architecture for Large Scale Virtual Environments*." Virtual Reality Annual International Symposium, 1995.

[17] M. Moreau. "*Documentation for the RTI*." George Mason University, 1997.

[18] K. L. Morse, L. Bic, M. Dillencourt, and K. Tsai. "*Multicast Grouping for Dynamic Data Distribution Management*," 31st Annual Summer Simulation Conf., 1999.

[19] K. L. Morse and J. S. Steinman, "*Data Distribution Management in the HLA: Multidimensional Regions and Physically Correct Filtering*," Spring Simulation Interoperability Workshop, 1997.

[20] M. J. Pullen, V. P. Laviano, and M. Moreau. "*Creating A Light-Weight RTI Using Selectively Reliable Transmission As An Evolution of Dual-Mode Multicast*." 97F-SIW-149, Fall Simulation Interoperability Workshop, 1997.

[21] S. J. Rak. "*Evaluation of Grid Based Relevance Filtering for Multicast Group Assignment*," 96-14-106, 14th Distributed Interactive Simulation Workshop, 1996.

[22] P. F. Reynolds, Jr., and S. Srinivasan. "*Communications, Data Distribution and Other Goodies in the HLA Performance Model*." 97S-SIW-050. Spring Simulation Interoperability Workshop, 1997.

[23] K. L. Russo, L.C. Schuette, J.E. Smith, and M.E. McGuire. "*Effectiveness of Various New Bandwidth Reduction Techniques in ModSAF*," 95-13-092. 13th DIS, 1995.

[24] Science Applications International Corporation (SAIC). "DMSO RTI Support Web Site ."

[25] J. S. Steinman et al. "*The SPEEDES-Based Run-Time Infrastructure for the High-Level Architecture on High-Performance Computers*." High-Performance Computing Conference, 1999, pp. 255-266.

[26] J. S. Steinman, Jeffrey S., T. Tran, J. Burckhardt, and J. S. Brutocao. "*Logically Correct Data Distribution Management in SPEEDES*." Technical Report, Metron Inc., 1999.

[27] J. S. Steinman, and F. Wieland. "*Parallel Proximity Detection and the Distribution List Algorithm*." Workshop on Parallel And Distributed Simulation (PADS), 1994, pp. 3-11.

[28] G. Tan, L. Xu, F. Moradi, and Y. Zhang. "*An Agent-based DDM Filtering Mechanism*." Eighth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems 2000, pp. 374-381.

[29] G. Tan, Y. Zhang, and R. Ayani."*A Hybrid Approach to Data Distribution Management*." IEEE Distributed Simulation and Real-Time Application, 2000, pp. 33-40.

[30] D. J. Van Hook, and J. O. Calvin. "*Data Distribution Management in RTI 1.3*." 98S-SIW-206, Spring Simulation Interoperability Workshop, 1998.

[31] D. J. Van Hook and S. J. Rak, James O. Calvin. "*Approaches to RTI Implementation of HLA Data Distribution Management Services*," 96-14-084, 15th DIS, 1996.